Project:
LUGSTAT

Equipo 4:

Jesus Lugo Santillán	A01089769	_____
Gonzalo Adrian Porras	A01281414	_____

Materia:
Diseño De Compiladores

Descripción del proyecto

Propósito, Objetivos y Alcance del Proyecto.

Proposito

El propósito de LUGSTAT es proveer un lenguaje de programación orientado a el análisis estadístico, manejo de matrices y vectores y simulación con múltiples funciones de análisis ya “built in” para hacer su uso fácil y conveniente.

Objetivos

Los objetivos del lenguaje dentro de su área (análisis de datos) son proveer un ambiente en cual se pueden hacer programas simples junto con el uso de sus funciones “built in” orientadas en el análisis de datos y manejo de matrices y vectores.

Alcance

El alcance es tener un ambiente funcional en cual se pueden programar funciones personalizadas para hacer cálculos y también dar uso de las funciones predefinidas que incluye el lenguaje.

Análisis de Requerimientos y Casos de Uso generales.

LUGSTAT, siendo un lenguaje de programación orientado a las matemáticas tiene que seguir ciertos lineamientos esperados de un lenguaje de programación moderno, como el manejo de diferentes tipos de variables básicas como Integers, Doubles y Booleans. También tiene que poder resolver expresiones matemáticas aritméticas como sumas, restas, divisiones y multiplicaciones; junto con las relacionales comunes como <, > , <=, >= , ==, !=

Todo lenguaje de programación debe de tener I/O básico junto con otros estatutos genéricos como asignaciones , ciclos y condicionales; por lo cual lugstat se espera que tenga estas habilidades por default, igual con la habilidad de poder manejar funciones definidas por el usuario con parámetros, sus propias variables etc..

Dado a que LUGSTAT es un lenguaje orientado a las matemáticas, se espera que tenga funciones “built in” de estadística, manejo de matrices y análisis de datos.

LUGSTAT puede ser usado para muchas cosas en particular, pero en general nosotros nos imaginamos que su uso cotidiano será para resolver y recolectar datos estadísticos de arreglos,

manejar matrices rápidamente para conseguir la inversa o transpuesta, etc. Como tiene la habilidad de hacer recursión, se abren las puertas para una gran cantidad de soluciones a problemas recursivos dando uso de LUGSTAT con el uso de sus funciones definidas por el usuario.

Descripción de los principales Test Cases.

Nosotros simplificamos mucho la situación de test cases en cada entrega dado a que a lo largo del proyecto fuimos generando test cases y los guardamos todos en un solo archivo para seguirlos probando mientras seguía avanzando el proyecto, de tal manera de asegurarnos que no rompieramos nada que previamente funcionaba bien. También se dio uso de varias de las hojas que nos entregó la maestra como test cases, metimos el código que se nos dio y si los cuádruplos salían iguales, era una manera fácil de asegurarse que algo complejo funciona rapidamente.

Descripción del PROCESO general seguido para el desarrollo del proyecto, incluyendo Bitácoras generales y un pequeño párrafo de reflexión de cada alumno, en relación a los principales aprendizajes logrados (firmarlo).

Ambos nosotros tenemos las mismas clases por lo cual teníamos tiempo para juntarnos en persona y discutir qué se iba a hacer y se pusieron metas cada semana. Fuera de clases nuestros horarios eran bastante diferentes dado a que uno de los miembros del equipo trabaja y el otro no, entonces uno trabajaba en el proyecto en la tarde y el otro continuaba durante la noche, de tal manera que la comunicación y notas de errores se hicieran claras para cuando cada uno termine alguna meta en particular. Cada actualización se subió a ramas separadas por miembro en el repositorio de GitHub y una vez confirmado el funcionamiento se unía con el master. Se trató de seguir el calendario que se nos proporcionó por la maestra los más posible como metas semanales.

Tratamos mucho de quedarnos al dia con el calendario que se nos proporciono pero por problemas de planeación del proyecto y falta de conocimiento de la herramienta que se usó nos terminamos atrasando un poco, pero con la "Semana i" y los asuetos logramos acercarnos a las metas originales propuestas por la maestra. - Gonzalo Garcia _____

Durante el desarrollo del proyecto se implementó la idea de llevarlo a cabo con el calendario entregado durante clase, aunque debido a falta de administración y otros proyectos hubo un retraso durante los primeros 3 avances, posteriormente entre la cuarta entrega y semana i logramos aprovechar el tiempo y ponernos de acuerdo para lograr concluir con el proyecto y estar al corriente con las fechas de entrega estimadas.

De manera personal el proyecto representó diversas dificultades y retos a lo largo del desarrollo, el primero de ellos fue crear la estructura de nuestro proyecto, y la manera de que lo íbamos a trabajar, nos decidimos por usar ply lex ya que lo habíamos utilizado durante la clase, una de las partes que considero que me fue de mucha ayuda fue desarrollar el proyecto con python, ya que tenía muy poco conocimiento y práctica, por lo que ha medida que avanzabamos fui poniendo en práctica conocimientos que había visto en materias anteriores, un ejemplo de esto fueron las estructuras de datos realizadas para controlar la memoria, el cubo semántico y las tablas de variables. En resumen fue un proyecto que me ayudó a trabajar diversas bases que fui aprendiendo a lo largo de la carrera.

- Jesus Lugo _____

Descripción del Lenguaje

Nombre del lenguaje

LUGSTAT

Descripción genérica de las principales características del lenguaje (en forma narrativa).

Por default LUGSTAT tiene un main, la sintaxis es la siguiente,
lugstat <nombre de tu main>;
{<codigo va aqui>}

El nombre de tu main puede incluir números y letras.

Para declarar variables globales, se contabilizan como si fueran las del main, y se agregan entre la declaración del main y el bloque del main. Lo mismo aplica para las variables locales de funciones, van entre la declaración de la función y su respectivo bloque. Los nombres de estas variables pueden contener letras y números.

Las funciones se declaran después de la declaración de variables globales, pero antes del bloque del main, su sintaxis es ***func <nombre> : <valor de retorno> (<variables> ;) {bloque}*** El valor de retorno de las funciones puede ser cualquiera de los tipos que maneja LUGSTAT, ya sea ***int, double, bool y void.***

La declaración de variables es ***var <nombre> (comma si se agrega otra) : tipo ;***

LUGSTAT **no** permite que se mezclen los tipos de variables durante su declaración por temas de organización, por lo cual variables de tipos diferentes tienen que estar en otra línea. Si se quieren múltiples variables del mismo tipo, nadamas se separan los nombres con comas. Se puede ver en este ejemplo:

```
lugstat mimain;
var int1, int2, int3: int ;
var bool1, bool2 : double;

Func mifunc : int (var param1: int; var param2 :double;)
var mifuncvar1, mifuncvar2 : int;
{
<codigo de mifunc va aqui>
}

{<codigo de main aqui>}
```

Para escribir a una variable se usa el comando read(X), donde X es el nombre de la variable donde se va a escribir el input
Para Imprimir existe la función print(X), donde X es el nombre de una variable, una expresión o un string. Strings están rodeados por comillas “asi”

Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución.

Variable doesn’t exist	Se hizo una operación con una variable que no existe en la tabla de variables
Function being summed does not exist	Se llamó a una función que no está en el directorio de funciones
Argument and Function parameter type	Se manaron argumentos de un tipo que no

Mismatch	coinciden con el tipo del parámetro de una función
Inconsistent number of arguments:parameters for function	Se mandaron más o menos argumentos de los que recibe una función como parámetros.
Illegal Character	Se ingresó un carácter ilegal.
Type mismatch	Se hizo una operación con tipos incompatibles en el cubo semántico.
E404	Funcion no existe
Function does not exist	Función en la cual se trato de agregar una variable no existe.
Variable already exists	Variable ya existe en tabla de variables que se trato de agregar
Syntax Error in input!	Existe un error de sintaxis.

Descripción del Compilador

Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.

El proyecto fue desarrollado en 3 sistemas operativos, todos con Python 3 Instalado.

- Windows 10 Education Build 1803
- Linux Mint 17.1 “Rebecca”
- Mac OS “Mojave”

El lenguaje que se usó es ***Python 3.***

Utilerias Especiales que se usaron fueron el módulo Queue que nadamas es para Python 3, PLY (LEX & YACC for Python) y el módulo Sys que se usó para hacer un tipo “HALT” cuando se llega a un error y que no continúe la compilación.

Descripción del Análisis de Léxico.

Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales.

Enumeración de los "tokens" del lenguaje y su código asociado.

t_PLUS = r'\+'	Esta expresion regular acepta +
t_MINUS = r'-'	Acepta -
t_MULT = r'*'	Acepta *
t_DIV = r'/'	Acepta /
t_PER = r'\%'	Acepta %
t_RELOP = r'==|<|>|<=|>=|!='	Acepta todos los operadores relacionales, <, > <= , >= y !=
t_GRE = r'<>'	Acepta Greater or Equal than <>
t_EQUALS = r'\='	Acepta =
t_OPAREN = r'\('	Acepta (
t_CPAREN = r'\)'	Acepta)
t_OBRACKET = r'\{'	Acepta {
t_CBRACKET = r'\}'	Acepta }
t_COLON = r':'	Acepta :
t_SCOLON = r';'	Acepta ;
t_COMMA = r','	Acepta ,
t_LCOR = r'\['	Acepta [
t_RCOR = r'\]'	Acepta]
t_QUOTE = r'\"'	Acepta “”

t_STRING = r'\".*\"'	Acepta “cualquier cosa como string con tal de que esta rodeada por comillas”
t_ignore_COMMENT = r'\#.*'	Esta expresión regular es usa para poder tener comentarios, ignora cualquier cosa que empieze con #
t_ignore = " \t"	Esta regex es para ignorar los tabs.
def t_newline(t): r'\n+' t.lexer.lineno += t.value.count("\n")	Esta regex se usa para contar líneas nuevas
def t_error(t): print("Illegal character '%s'" % t.value[0]) t.lexer.skip(1)	Este token marca errores, en caso de valores extraños o no permitidos
def t_ID(t): r'[a-zA-Z_][a-zA-Z_0-9]*' t.type = reserved.get(t.value,'ID') # Check for reserved words return t	Este token identifica los ID’s. pueden tener letras o números.

Descripción del Análisis de Sintaxis.

Gramática Formal empleada para representar las estructuras sintácticas (Sin “codificar”).

Main
LUGSTAT es el token de inicio para lugstat.
ID es el nombre de el main
SCOLON es ;
Lugstat2 es la función que va a las variables
Lugstat3 es la función que va a los módulos
	`lugstat : LUGSTAT ID SCOLON lugstat2 lugstat3 block`
	
	`lugstat mimain;`
	`var mivar1 : int;`
	`func mifunc : int (param1: int;) {}`
	`{}`

Variables
“vars1” es la gramática para definir variables, brinca desde la regla vars donde esta el token “var” asign2 llama a la regla de declaracion de arreglos y matrices.
	`vars : VAR vars1`
	
	`vars1 : ID COMMA vars1`
	`| ID COLON tipo SCOLON lugstat2`
	`| ID asign2 COLON tipo SCOLON`
	`| ID asign2 COMMA vars1`
	
	`var mivar1 : int;`

LUGSTAT solo permite declaraciones genéricas. Estas declaraciones se pueden juntar en una sola con commas si son del mismo tipo.	`var mibool1, mibool2 :bool;` `var midouble1, midobule2 :double;`

Funciones
 	
Modules es la declaración de los módulos o funciones adicionales. Inicia con FUNC
ID es el nombre del módulo
Tipo es el tipo de retorno
 	
modules2 manda a llamar la declaración de variables, primero para parámetros luego las variables locales de la función.
 	
Funblock es el bloque de la función.
	`modules : FUNC ID COLON tipo OPAREN`
	`modules2 CPAREN modules2 funblock`
	
	`func mifunc : int (param1: int;)`
	`var mivlocal1: int;`
	`{}`

Bloques
 	
Los bloques son estatutos rodeados por brackets.
 	
Dentro de los estatutos están todas las funciones y operaciones.
	`block : OBRACKET block2 CBRACKET`
	
	`'''`
	`block2 : estatuto`
	`| estatuto block2`
	`| empty'''`

Estatutos basicos
 	
Estos son los estatutos básicos, cada uno manda a su regla.
Entre ellos están el de asignación, las condiciones, estructura, do while, llamada a función y al read line.
	`'''`
	`estatuto : asign`
	`| cond`
	`| escrt`
	`| plot`
	`| count`
	`| countif`
	`| metodos`
	`| dwhile`
	`| readln`
	`| funccall`
	`'''`

Asignacion
 	
LUGSTAT permite varios tipos de asignación, ya sea el resultado de una expresión, igualar variables, igualar el contenido de un arreglo y operaciones de arreglos en general.
	`'''`
	`asign : ID EQUALS expresion SCOLON`
	`| ID EQUALS ID SCOLON`
	`| ID EQUALS ID asign2 SCOLON`
	`| ID asign2 EQUALS ID SCOLON`
	`| ID assign2 EQUALS expresion SCOLON`
	`| ID asign2 EQUALS ID asign2 SCOLON`
	`'''`
	
	`Var1 = 1+1*(100/10);`
	`Var1 = Var2;`
	`Var1 = Arr1[1];`
	`Arr1[2+1] = Arr2[3+i];`
	`. . .`
	`Arr1[2] = 1*3;`

Escritura
 	
El estatuto de escritura de LUGSTAT escrt permite juntar predicados con commas y imprimirlos, ya sean variables, expresiones o strings.
	'''escrt : PRINT OPAREN ID escrt2 CPAREN SCOLON
	| PRINT OPAREN expresion CPAREN SCOLON
	| PRINT OPAREN STRING CPAREN SCOLON
	'''
	
	Print (var1);
	print(var1, var2);
	print(1+1);
	print("hola mundo!");

Condiciones
 	
LUGSTAT soporta estatutos condicionales tipo IF y IF - ELSE para expresiones con operadores relacionales == , < ,> ,<= ,>= ,!=
	'''cond : IF OPAREN expresion CPAREN ifblock SCOLON
	| IF OPAREN expresion CPAREN ifblock ELSE ifblock SCOLON
	'''
	
	If (a > b) {};
	. . .
	If (a > b) {} else {};

Expresiones El manejo de expresiones de LUGSTAT está basado en el de littleduck2019.	'''expresion : exp | expresion RELOP exp ''' '''exp : termino | termino PLUS exp | termino MINUS exp ''' '''termino : factor | factor MULT termino | factor DIV termino ''' '''factor : OPAREN expresion CPAREN | varcte ''' '''varcte : ID | ID asign2 | NUMERIC | NUMBER '''
Do-While La manera que hace lugstat sus ciclos es usando DO - WHILE	'''dwhile : DO wblock WHILE OPAREN dwhileconds CPAREN SCOLON '''
Read Line	''' readln : READ OPAREN ID CPAREN SCOLON

LUGSTAT soporta inputs por el usuario usando la función read, lee y asigna a la variable lo que se leyó.	''' read(k);

Descripción de Generación de Código Intermedio y Análisis Semántico. Debe incluir:

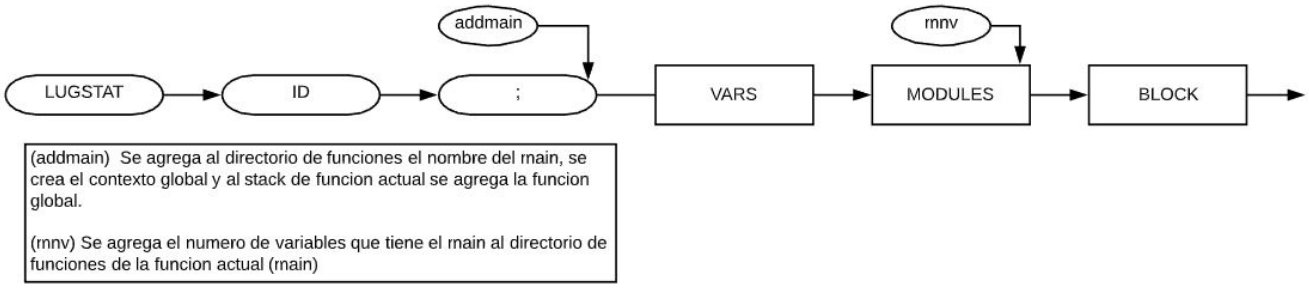
1.

Código de operación y direcciones virtuales asociadas a los elementos del código

Nosotros no usamos códigos de operación, LUGSTAT maneja los cuádruplos de una manera fácilmente comprensible con su nombre, Los tipos de datos se separaron en diferentes segmentos de memoria.	

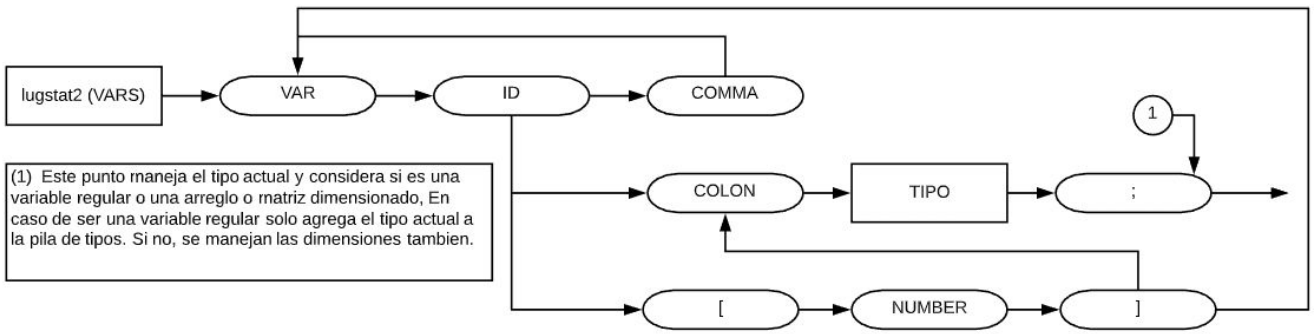
#Globales
Ginteger = 0
Gdouble = 2500
Gbool = 5000
Gstring = 7500
#Locales
Linteger = 10000
Ldouble = 12500
Lbool = 15000
Lstring = 17500
#Temporales
Tinteger = 20000
Tdouble = 22500
Tbool = 25000
Tstring = 27500
#Constantes
Cinteger = 30000
Cdouble = 32500
Cbool = 35000
Cstring = 37500
	Cada tipo de variable tiene su propio “segmento” de memoria, de tal manera que se independiza de las demás.
	
	Se separaron por tipo de variable, y por tipo de datos, las Integer, Double, Boolean y Float tienen su propio rango en memoria.

Diagramas de Sintaxis con las acciones correspondientes.

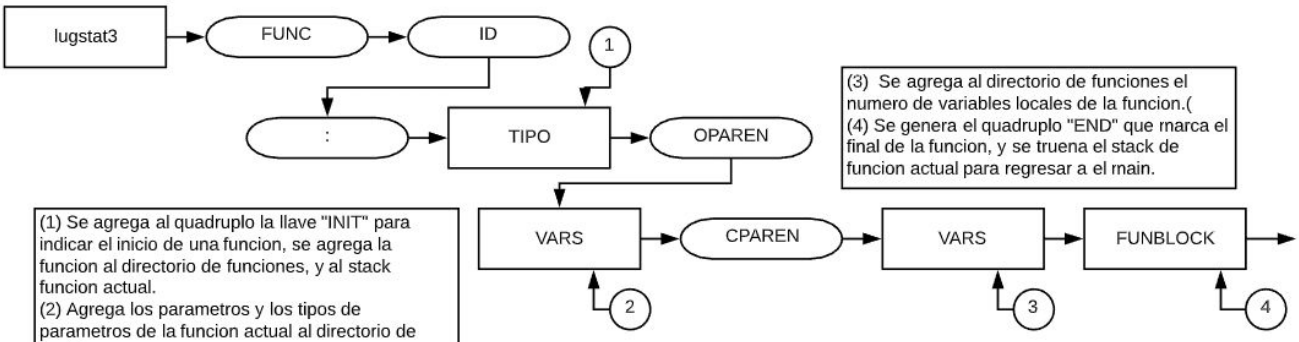


(addmain) Se agrega al directorio de funciones el nombre del rnain, se crea el contexto global y al stack de funcion actual se agrega la funcion global.

(rnnv) Se agrega el numero de variables que tiene el rnain al directorio de funciones de la funcion actual (rnain)

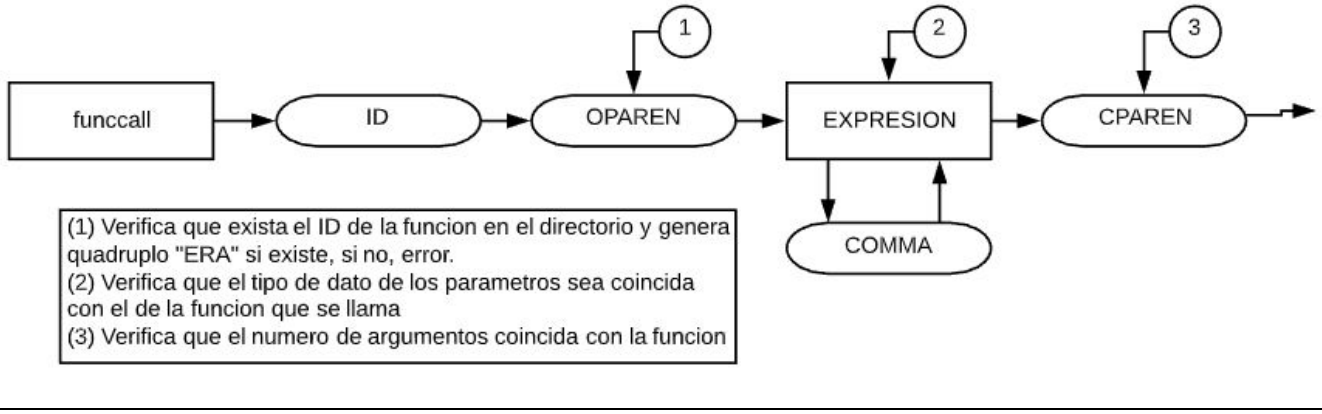


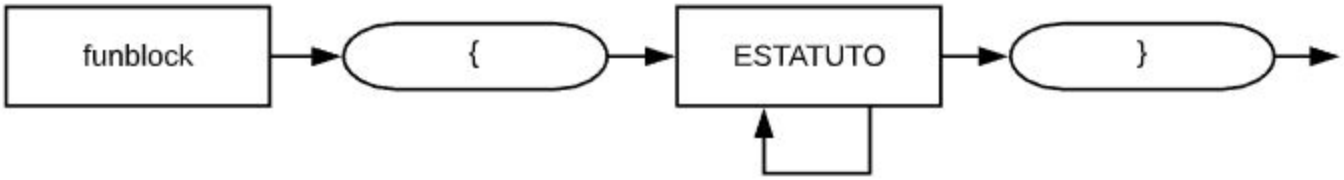
(1) Este punto rnaneja el tipo actual y considera si es una variable regular o una arreglo o rnatriz dimensionado, En caso de ser una variable regular solo agrega el tipo actual a la pila de tipos. Si no, se manejan las dimensiones tambien.

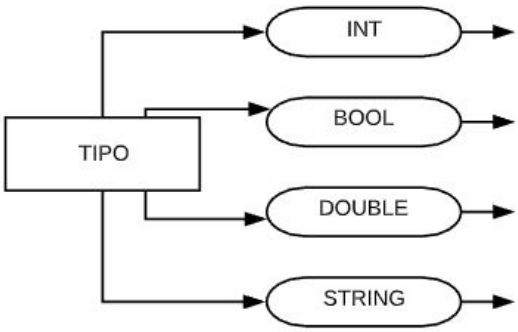


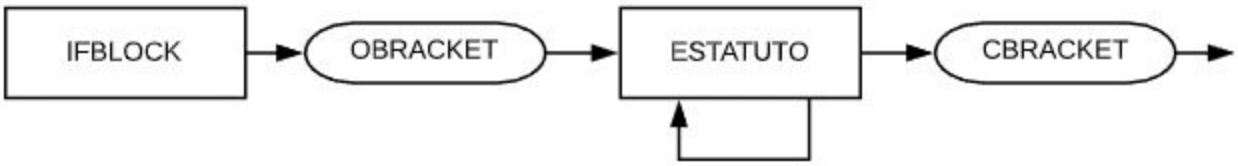
(1) Se agrega al quadruplo la llave "INIT" para indicar el inicio de una funcion, se agrega la funcion al directorio de funciones, y al stack funcion actual.
(2) Agrega los parametros y los tipos de parametros de la funcion actual al directorio de funciones.

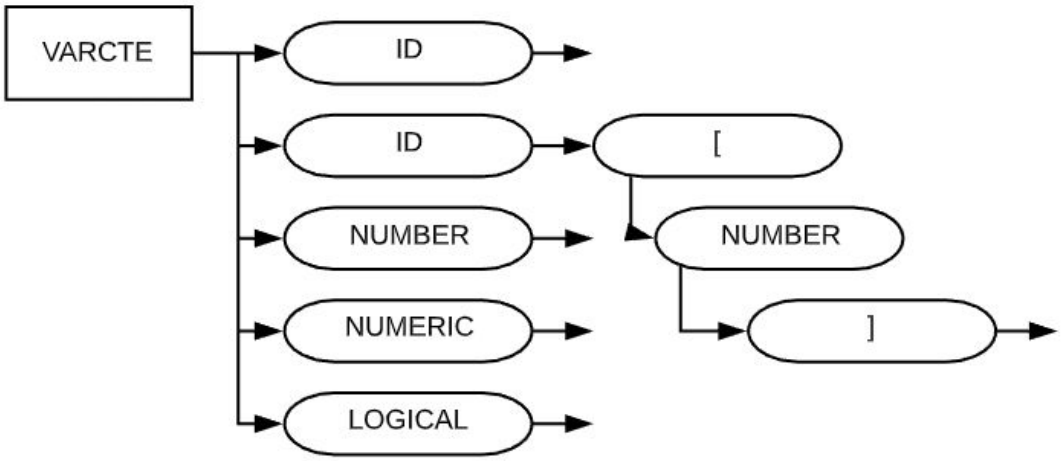
(3) Se agrega al directorio de funciones el numero de variables locales de la funcion.(
(4) Se genera el quadruplo "END" que rnarca el final de la funcion, y se truena el stack de funcion actual para regresar a el rnain.

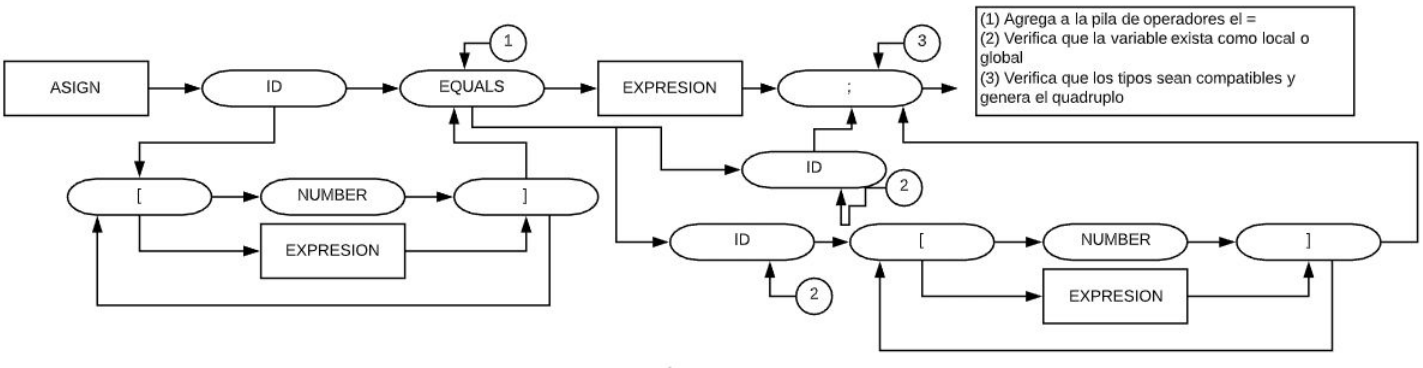

funccall
ID
OPAREN
EXPRESION
CPAREN
COMMA
1
2
3
(1) Verifica que exista el ID de la funcion en el directorio y genera quadruplo "ERA" si existe, si no, error.
(2) Verifica que el tipo de dato de los parametros sea coincida con el de la funcion que se llama
(3) Verifica que el numero de argumentos coincida con la funcion

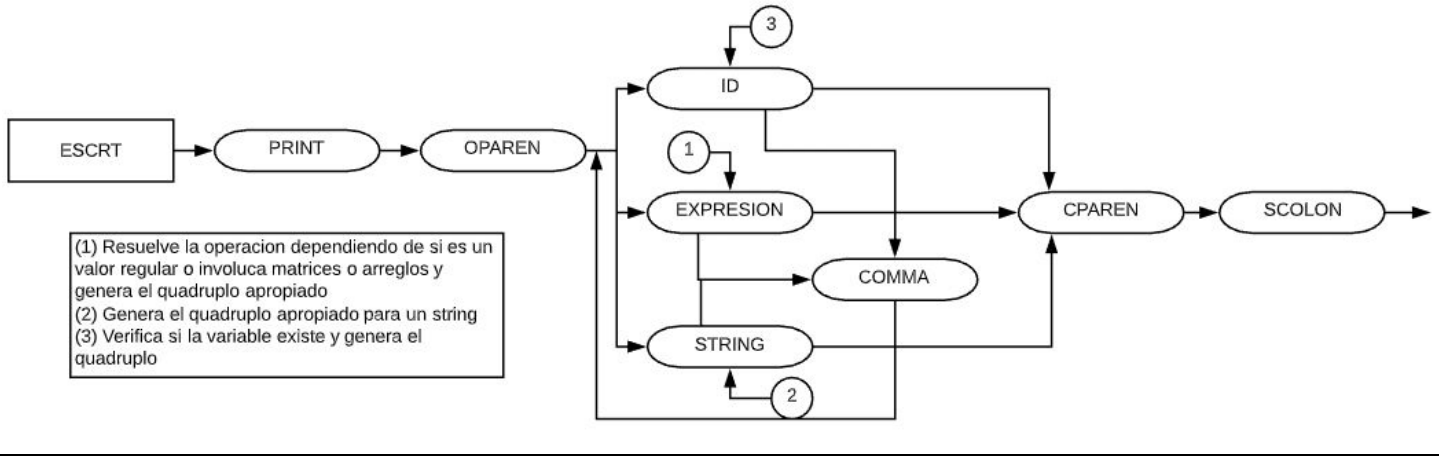

funblock
{
ESTATUTO
}

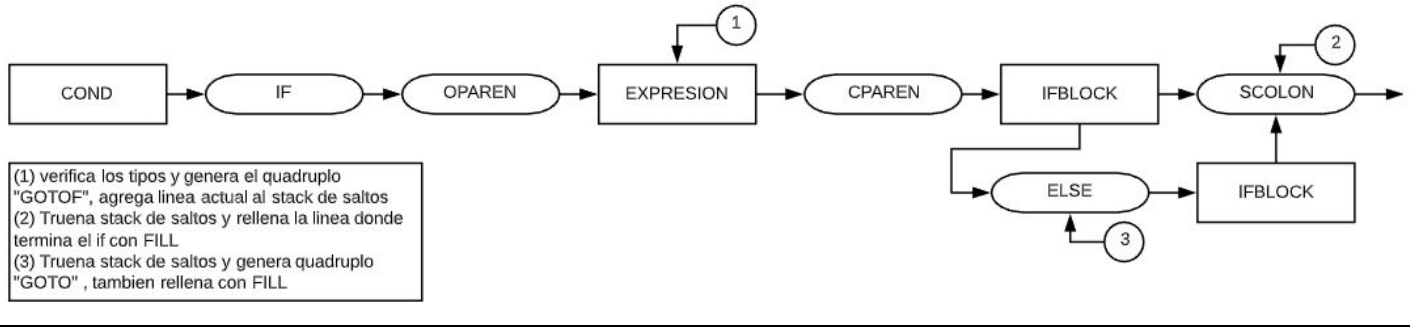

TIPO
INT
BOOL
DOUBLE
STRING

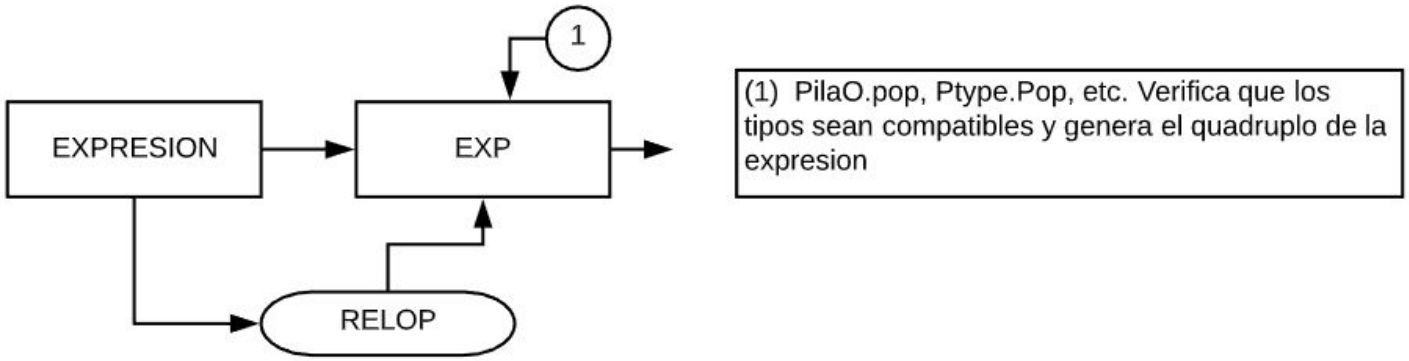

IFBLOCK
OBRACKET
ESTATUTO
CBRACKET

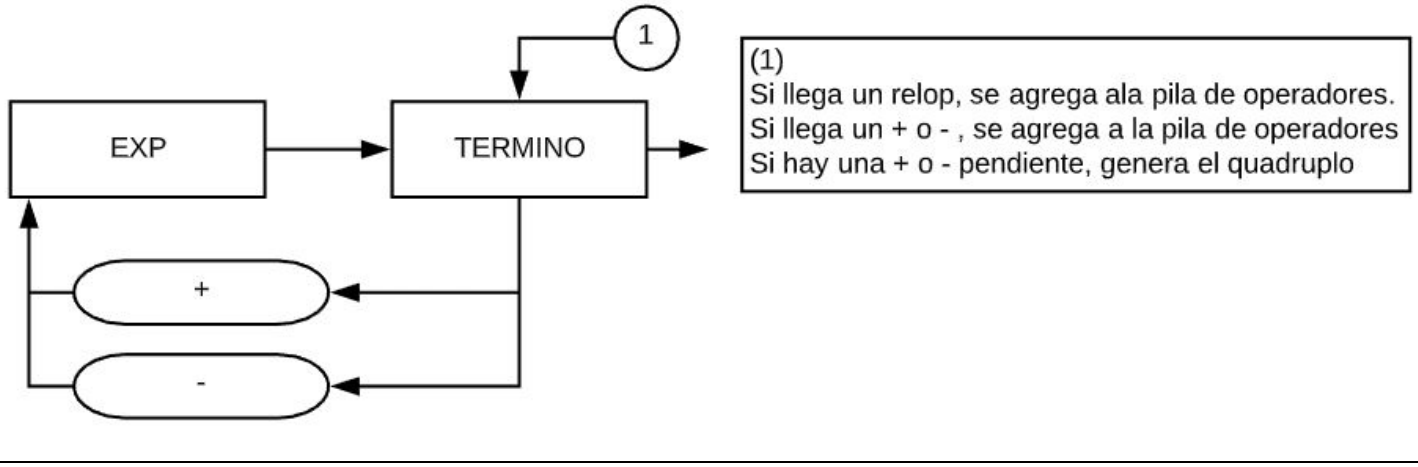

VARCTE
ID
ID
[
NUMBER
NUMBER
NUMERIC
]
LOGICAL

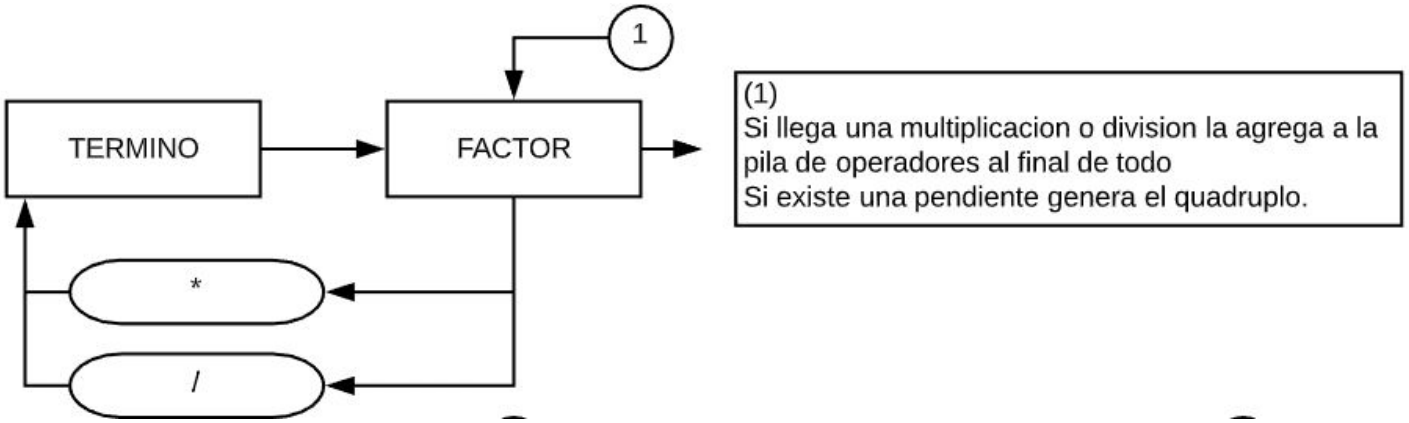

ASIGN
ID
EQUALS
EXPRESION
;
1
3
(1) Agrega a la pila de operadores el =
(2) Verifica que la variable exista como local o global
(3) Verifica que los tipos sean compatibles y genera el quadruplo
[
NUMBER
]
EXPRESION
ID
2
ID
[
NUMBER
]
EXPRESION
2

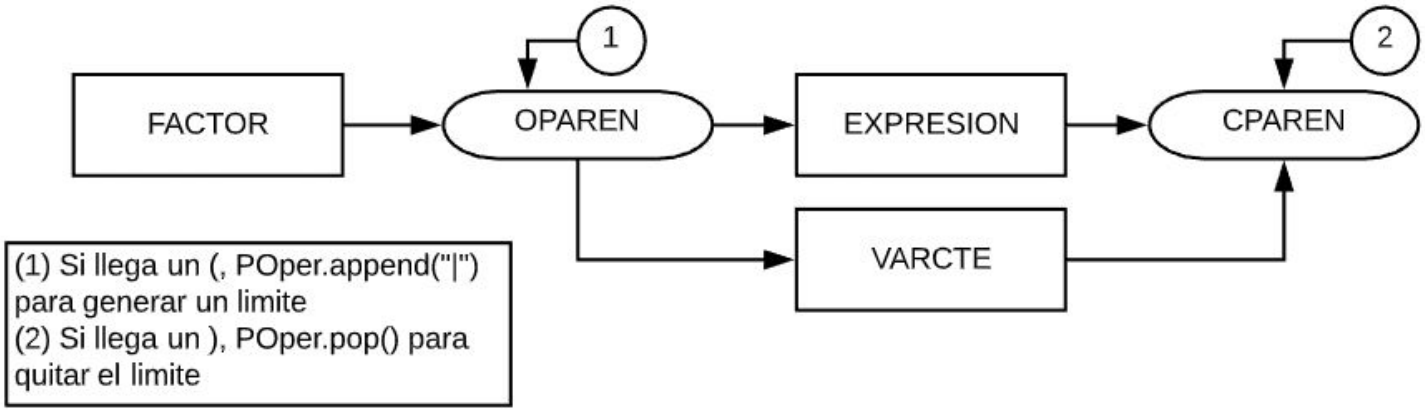

ESCRT
PRINT
OPAREN
3
ID
1
EXPRESION
CPAREN
SCOLON
COMMA
STRING
2
(1) Resuelve la operacion dependiendo de si es un valor regular o involuca matrices o arreglos y genera el quadruplo apropiado
(2) Genera el quadruplo apropiado para un string
(3) Verifica si la variable existe y genera el quadruplo

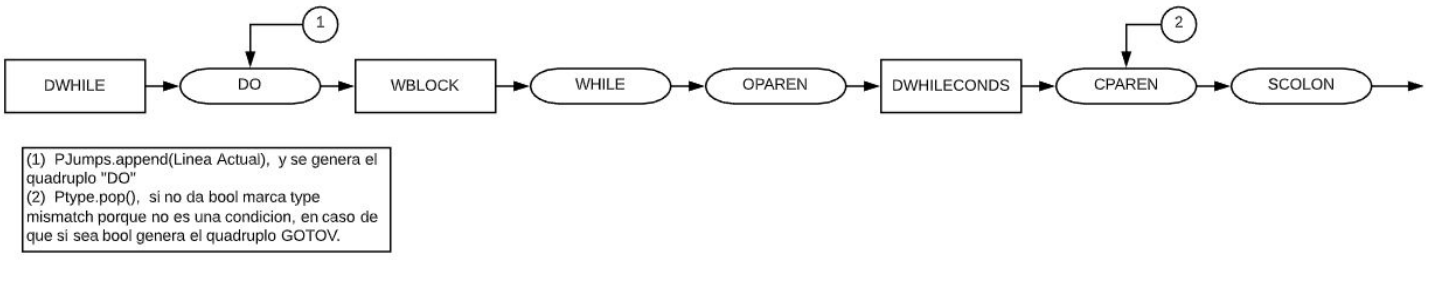

COND
IF
OPAREN
EXPRESION
CPAREN
IFBLOCK
SCOLON
ELSE
IFBLOCK
1
2
3
(1) verifica los tipos y genera el quadruplo "GOTOF", agrega linea actual al stack de saltos
(2) Truena stack de saltos y rellena la linea donde termina el if con FILL
(3) Truena stack de saltos y genera quadruplo "GOTO" , tambien rellena con FILL

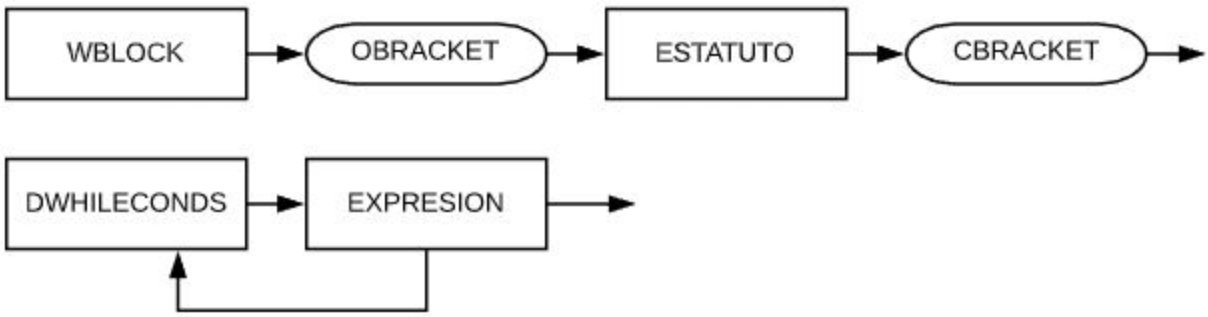

EXPRESION
EXP
RELOP
1
(1) PilaO.pop, Ptype.Pop, etc. Verifica que los tipos sean compatibles y genera el quadruplo de la expresion

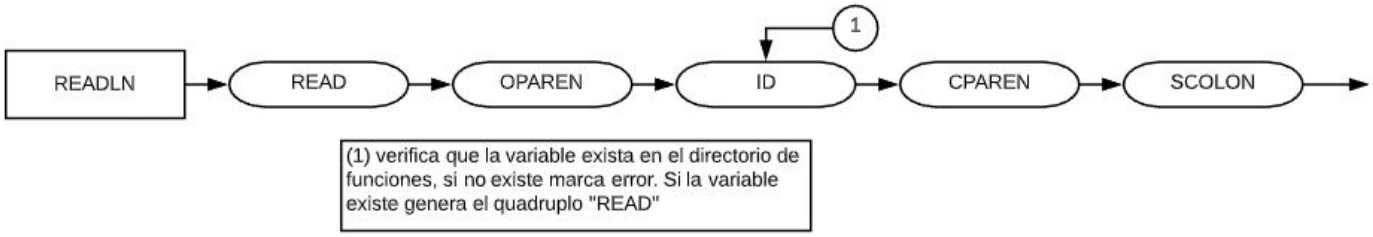

EXP
TERMINO
+
-
1
(1)
Si llega un relop, se agrega ala pila de operadores.
Si llega un + o - , se agrega a la pila de operadores
Si hay una + o - pendiente, genera el quadruplo

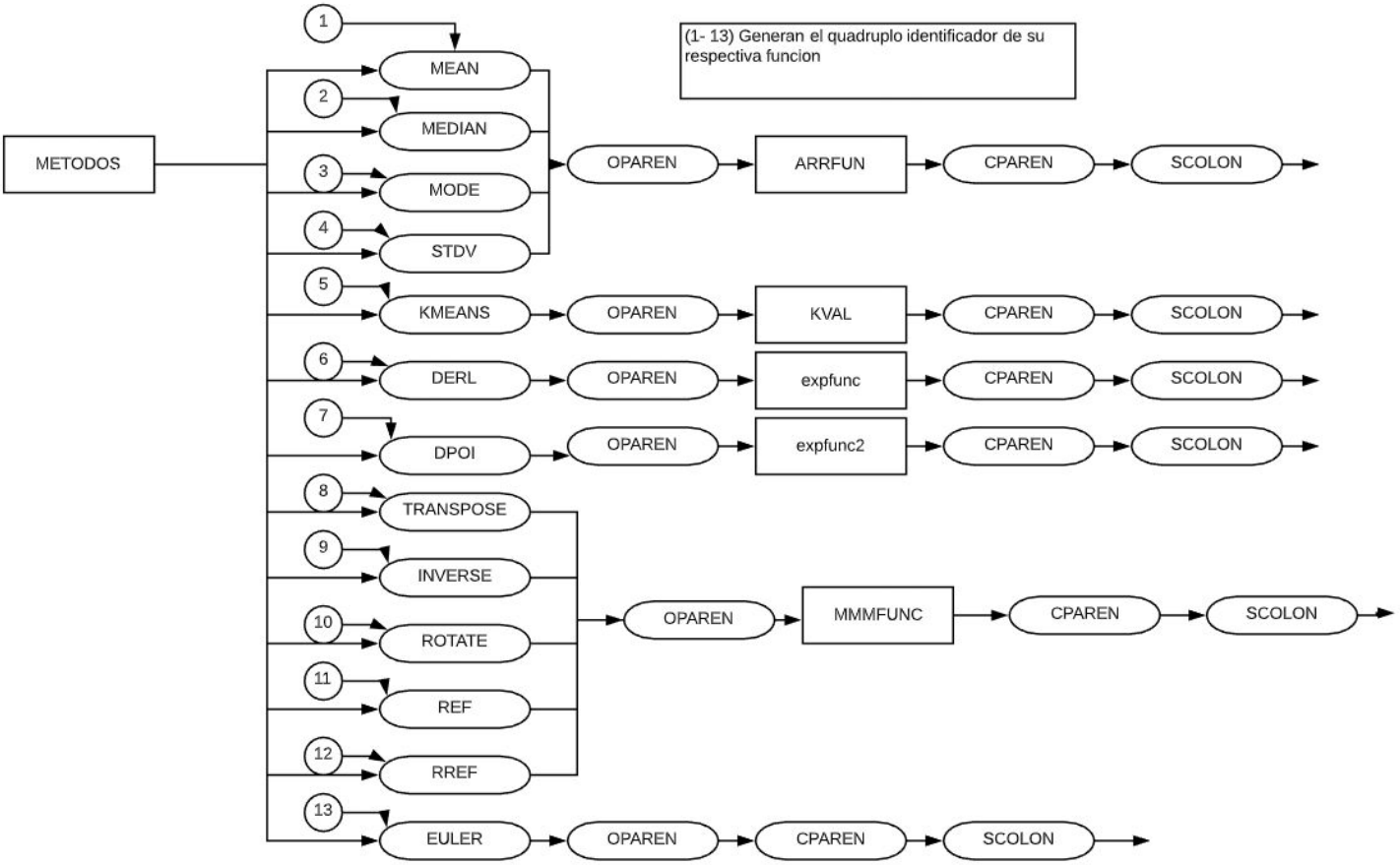

1
TERMINO
FACTOR
*
/
(1)
Si llega una multiplicacion o division la agrega a la pila de operadores al final de todo
Si existe una pendiente genera el quadruplo.

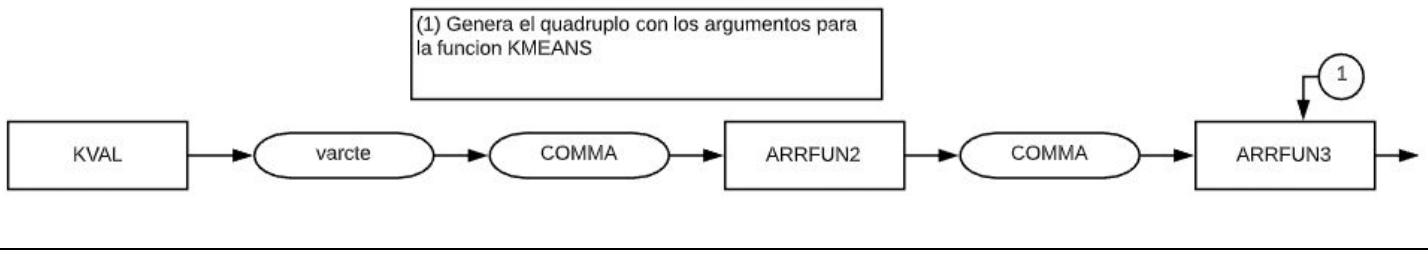

1
2
FACTOR
OPAREN
EXPRESION
CPAREN
VARCTE
(1) Si llega un (, POper.append("|") para generar un limite
(2) Si llega un), POper.pop() para quitar el limite

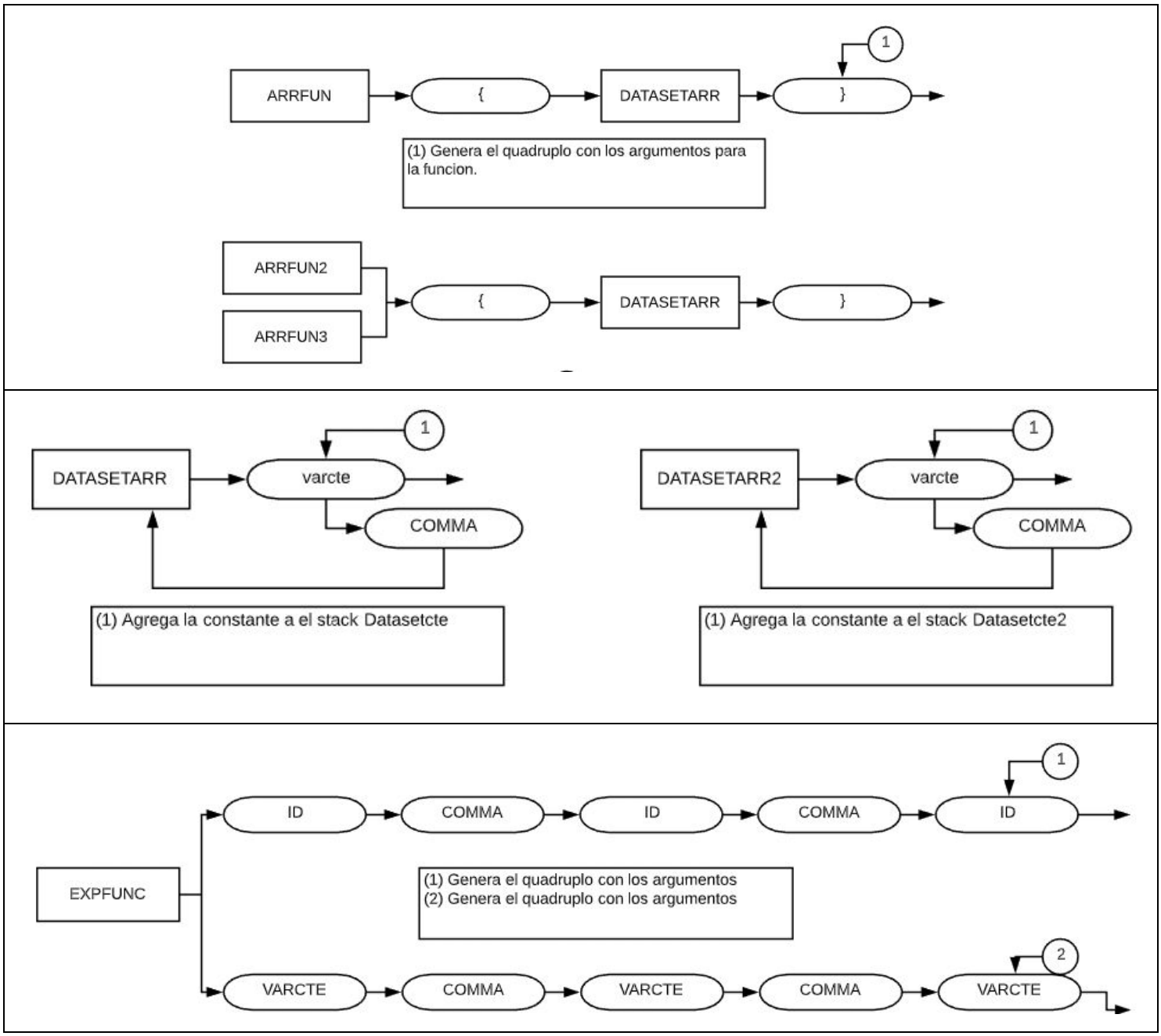

1
2
DWHILE
DO
WBLOCK
WHILE
OPAREN
DWHILECONDS
CPAREN
SCOLON
(1) PJumps.append(Linea Actual), y se genera el quadruplo "DO"
(2) Ptype.pop(), si no da bool marca type mismatch porque no es una condicion, en caso de que si sea bool genera el quadruplo GOTOV.

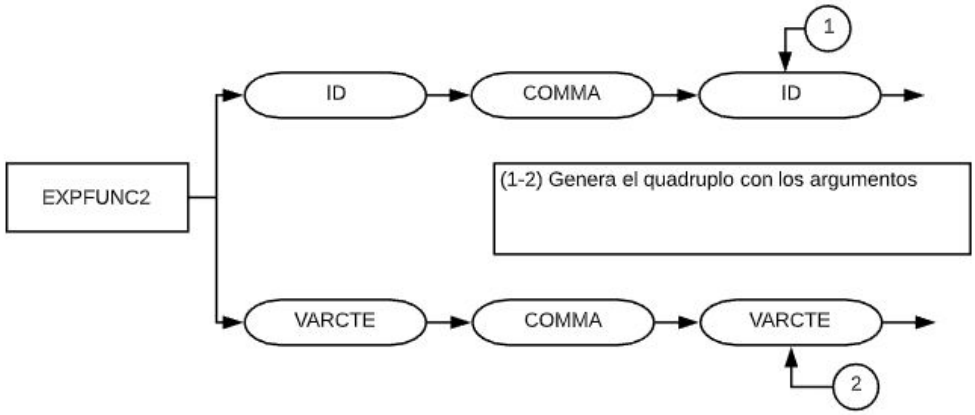

WBLOCK
OBRACKET
ESTATUTO
CBRACKET
DWHILECONDS
EXPRESION

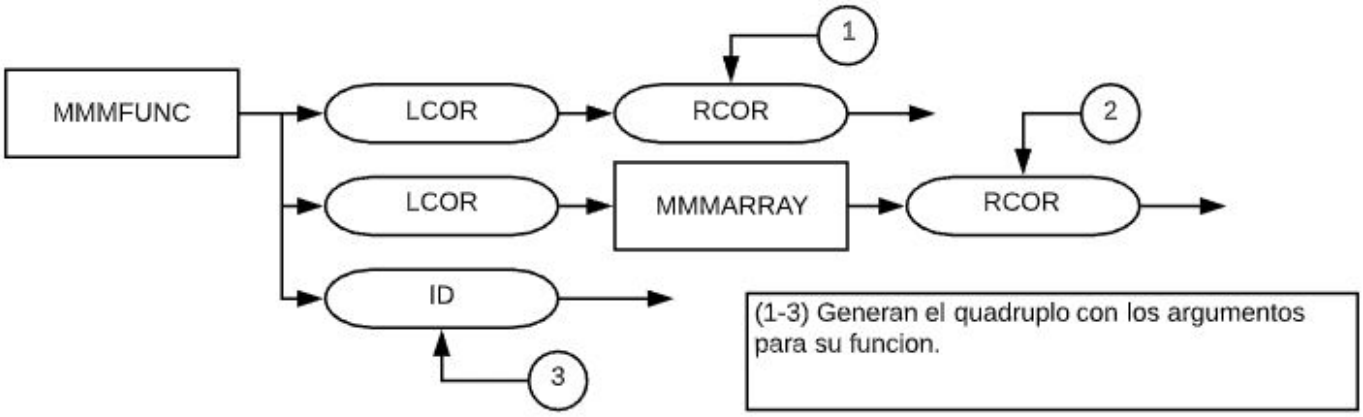

READLN
READ
OPAREN
1
ID
CPAREN
SCOLON
(1) verifica que la variable exista en el directorio de funciones, si no existe marca error. Si la variable existe genera el quadruplo "READ"

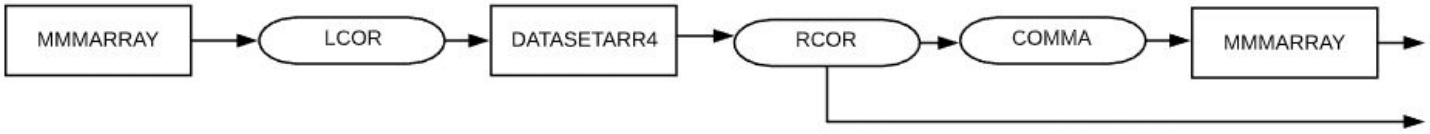

METODOS
1
MEAN
2
MEDIAN
3
MODE
4
STDV
5
KMEANS
6
DERL
7
DPOI
8
TRANSPOSE
9
INVERSE
10
ROTATE
11
REF
12
RREF
13
EULER
OPAREN
ARRFUN
CPAREN
SCOLON
KVAL
expfunc
expfunc2
MMMFUNC
(1- 13) Generan el quadruplo identificador de su respectiva funcion

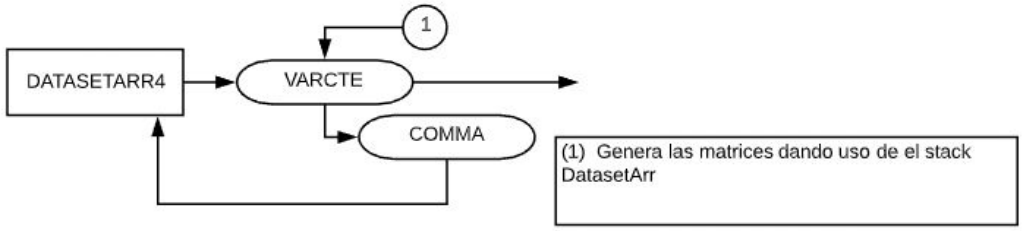

KVAL
varcte
COMMA
ARRFUN2
COMMA
1
ARRFUN3
(1) Genera el quadruplo con los argumentos para la funcion KMEANS


ARRFUN
{
DATASETARR
}
1
(1) Genera el quadruplo con los argumentos para la funcion.
ARRFUN2
ARRFUN3
{
DATASETARR
}
DATASETARR
varcte
COMMA
1
(1) Agrega la constante a el stack Datasetcte
DATASETARR2
varcte
COMMA
1
(1) Agrega la constante a el stack Datasetcte2
EXPFUNC
ID
COMMA
ID
COMMA
ID
1
(1) Genera el quadruplo con los argumentos
(2) Genera el quadruplo con los argumentos
VARCTE
COMMA
VARCTE
COMMA
VARCTE
2


1
ID
COMMA
ID
EXPFUNC2
(1-2) Genera el quadruplo con los argumentos
VARCTE
COMMA
VARCTE
2


1
MMMFUNC
LCOR
RCOR
2
LCOR
MMMARRAY
RCOR
ID
(1-3) Generan el quadruplo con los argumentos para su funcion.
3


MMMARRAY
LCOR
DATASETARR4
RCOR
COMMA
MMMARRAY


1
DATASETARR4
VARCTE
COMMA
(1) Genera las matrices dando uso de el stack DatasetArr

Breve descripción de cada una de las acciones semánticas y de código (no más de 2 líneas).

Nuestras consideraciones semánticas funcionan haciendo uso de la `get_tipo(self, Izq, Der, operador)`, en donde se evalúan los tipos y en base a la operación regresa un resultado específico.

Tabla de consideraciones semánticas.

```
self.Consideracion = {
            #INT
            'int' : { 'int': { #Entero y Entero
               '+' : 'int', '-' : 'int', '*' : 'int', '/' : 'int', '%' : 'int', '=' : 'int',
               '==' : 'bool', '<' : 'bool', '>' : 'bool', '<=' : 'bool', '>=' : 'bool','!=' : 'bool','<>' : 'bool',
            },
            'double' : { #Entero y doble
               '+' : 'double', '-' : 'double', '*' : 'double', '/' : 'double', '%' : 'int', '=' : 'int',
               '==' : 'bool', '<' : 'bool', '>' : 'bool', '<=' : 'bool', '>=' : 'bool','!=' : 'bool','<>' : 'bool',
            },
            'bool' : { #Entero y bool
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            },
            'char' : { #Entero y bool
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            },
            'string' : { #Entero y bool
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            }
             },

            #DOUBLE
             'double' : { 'int': { #doble y Entero
               '+' : 'double', '-' : 'double', '*' : 'double', '/' : 'double', '%' : 'double', '=' : 'double',
               '==' : 'bool', '<' : 'bool', '>' : 'bool', '<=' : 'bool', '>=' : 'bool','!=' : 'bool','<>' : 'bool',
            },
            'double' : { #doble y doble
               '+' : 'double', '-' : 'double', '*' : 'double', '/' : 'double', '%' : 'double', '=' : 'double',
               '==' : 'bool', '<' : 'bool', '>' : 'bool', '<=' : 'bool', '>=' : 'bool','!=' : 'bool','<>' : 'bool',
            },
            'bool' : { #doble y bool
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
```

```
},
'char' : { #doble y char
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'string' : { #doble y string
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
}
 },

#BOOL
 'bool' : { 'int': { #bool y Entero
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'double' : { #bool y doble
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'bool' : { #bool y bool
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'bool',
   '==' : 'bool', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'bool','<>' : 'error',
},
'char' : { #bool y char
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'string' : { #bool y string
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
}
 },

#CHAR
 'char' : { 'int': { #CHAR y Entero
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'double' : { #CHAR y doble
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
'bool' : { #CHAR y bool
   '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
   '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
},
```

```
            'char' : { #CHAR y CHAR
               '+' : 'char', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'char',
               '==' : 'bool', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'bool','<>' : 'error',
            },
            'string' : { #CHAR y String
               '+' : 'string', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
#Revisar == y != como bools ?
            }
             },

            #String
             'String' : { 'int': { #String y Entero
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            },
            'double' : { #String y doble
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            },
            'bool' : { #String y bool
               '+' : 'error', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'error', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'error','<>' : 'error',
            },
            'char' : { #String y char
               '+' : 'string', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'error',
               '==' : 'bool', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'bool','<>' : 'error',
#Revisar == y != como bools ?
            },
            'string' : { #String y String
               '+' : 'string', '-' : 'error', '*' : 'error', '/' : 'error', '%' : 'error', '=' : 'string',
               '==' : 'bool', '<' : 'error', '>' : 'error', '<=' : 'error', '>=' : 'error','!=' : 'bool','<>' : 'error',
            }
             }

        }
```

Descripción detallada del proceso de Administración de Memoria usado en la compilación.

En compilación, se generan las direcciones de memoria automáticamente en la declaración de una variable, en el mismo momento que se registra en la tabla de variables actual. Como se mencionó previamente; se hace un chequeo de qué tipo de variable es para poder asignarle el número de segmento de memoria apropiado, también depende el "contexto" en el que va a estar. Las variables globales se les asignan diferentes segmentos de memoria que a el resto, y también se separan por el tipo de la variable. En resumen; se verifica el tipo y se asigna la dirección de memoria que tiene en base de su "scope" y el tipo de dato que contendrá la variable.

Durante la compilación se genera automáticamente también el primer contexto, que es el contexto global. Este simplemente se usa como base para las direcciones, y el número de direcciones ocupadas se resetea cuando se llega a una definición de una función.

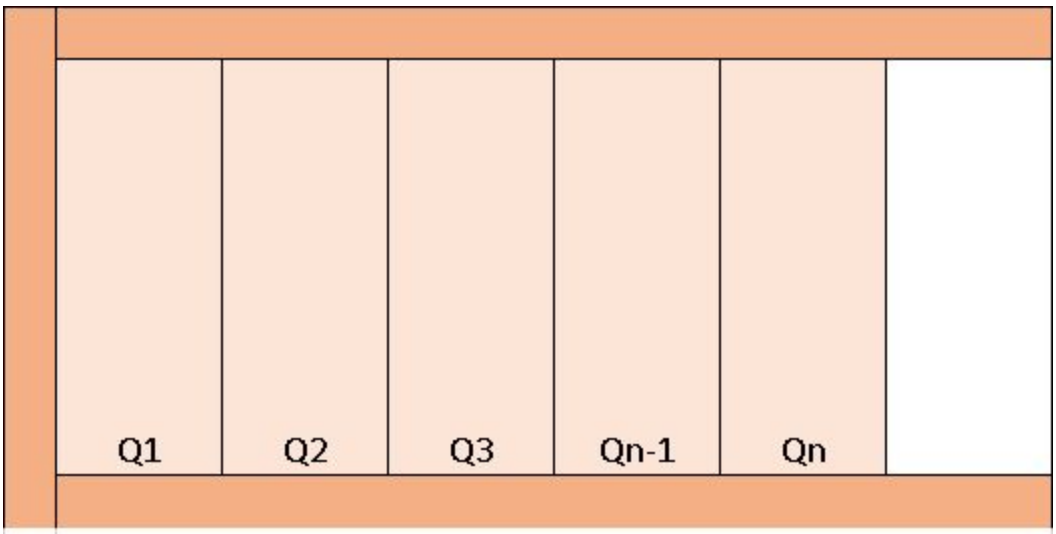
Especificación gráfica de CADA estructura de datos usada (Dir.Funciones, Tablas de Var's, Cuádruplos, etc...)

DIRECTORIO DE FUNCIONES						
NOMBRE	TIPO RETORNO	LINEA DE INICIO	# DE PARAMS	# DE VARIABLES	TIPO PARAMS	VARS
Main	None	1	None	1	Int	TVMain
Func1	Int	4	1	0	Int	TVFunc1
Func2	double	10	2	1	double	TVFunc2
...	...	...	...	...	...	...
Funcn-1	Int	15	1	1	Int	TVFuncn-1
Func n	Int	20	1	1	Int	TVFuncn

El directorio de funciones es un diccionario, gráficamente lo tenemos descrito aquí como una tabla para simplicidad. Se usó un diccionario gracias a su fácil acceso y la facilidad de agregar elementos en caso de que se necesiten.

TABLA DE VARIABLES		
NOMBRE	TIPO	UBICACION EN MEMORIA
int1	Int	10000
int2	Int	10001
int3	Int	10002
...	...	...
double1	double	12500

La tabla de variables también es un diccionario por su facilidad de acceso, nosotros lo representamos como una tabla para su fácil comprensión.



Q1			
+	1	1	T1
Q2			
+	1	T1	T2
Q3			
=	T2		A
Q4			
Read			B

Para nuestros cuádruplos nosotros utilizamos una fila, de la librería Queue de Python3. Esto nos permite tener todo organizado y con fácil acceso a la lista, junto con la habilidad de “tronar” el primero en la fila y que automáticamente continúe con el siguiente.

Arreglos dentro de tabla de variables				
Nombre	Tipo	Inicio	Final	Dimension
Arr1	Int	10000	10005	0
Arr2	Double	12500	12505	0
Mat1	Int	10006	10010	5

La tabla de variables también se adapta a la declaración de arreglos en dónde los arreglos cuentan con los atributos extras de Inicio, final y dimensión.

Descripción de la Máquina virtual

Equipo de cómputo, lenguaje y utilerías especiales usadas (en caso de ser diferente que el compilador).

El proyecto fue desarrollado en 3 sistemas operativos, todos con Python 3 Instalado.

- Windows 10 Education Build 1803
- Linux Mint 17.1 “Rebecca”
- Mac OS “Mojave”

El lenguaje que se usó es ***Python 3.***

Entre dependencias extras instaladas mediante el comando pip se encuentra.

- Numpy
- Pandas
- Matplotlib
- Sklearn
- Sklearn

Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura)

Asociación entre direcciones virtuales en compilación y reales en ejecución

```
def createMemory(self): #      G              L              T              C
    self.memoria =      [[{},{},{},{}], [[{},{},{},{}]],[[{},{},{},{}]],[{},{},{},{}]]
```

La memoria la controlamos de la siguiente manera, primero se divide en niveles, en donde cada casilla representa el tipo de direcciones, por otro lado el apartado de memoria y local es un arreglo, ya que pueden existir diversas memorias locales y temporales, por función.
Para acceder y agregar un valor se hace uso de la siguiente formula, en este caso la dirección la manejas de maner absoluta, ya que debido a accesos indirectos como los vistos en clase evitar problemas con cualquier número negativo, posteriormente se define el nivel dividiendo entre 10,000 ya que las direcciones globales,locales,temporales, y constantes cuentan con 10,000 espacios de memoria asignados 2,500 para cada tipo de variable aceptada, en el segundo caso se decide el tipo haciendo uso del módulo de la dirección entre la cantidad de memoria asignada, y por último se asigna el index dentro de la memoria.

```
def addMemoryValue (self,direccion,valor):
        if abs(direccion) // 10000 == 0: #Guardamos valor en Globales
            self.memoria[0][((abs(direccion) % 10000) //
2500)][(abs(direccion) % 10000) % 2500] = valor # [Nivel es decir GLTC]
[Tipo de variable][Index de variable]
```

En ejecución, se generan los contextos cuando se hace una llamada a una función, por default está el contexto del main que se generó en compilación. Los contextos se generan cuando se llega a un quadruplo “ERA”, y se hace una llamada a **memory.createLocalTemporal().** Una vez que una función termina se elimina su contexto y regresa al contexto anterior. Esto permite que fácilmente se haga recursión fácilmente sin problemas. Los contextos se eliminan con la llamada al siguiente módulo de memoria: **memory.freeFunctionMemory()**

```
    def freeFunctionMemory(self):
        self.memoria[1].pop() #Al terminar una funcion eliminamos las
variables locales
        self.memoria[2].pop() #Al terminar una funcion eliminamos las
variables Temporales

    def createLocalTemporal(self):
```

```
        self.memoria[1].append([{},{},{},{}]) #Al crear una funcion
siempre crear una nueva direccion
        self.memoria[2].append([{},{},{},{}]) #Al crear una funcion
eliminamos las variables Temporales
```

En este caso buscamos a qué contexto pertenece una locación, en caso de que no se encuentre en el contexto actual, se busca en el contexto anterior, siempre hay prioridad al elegir el contexto actual

```
     try:
          addrv = memory.getActualContextValue(addr+ActualQ[2])
                                print(addrv)
     except KeyError:
          addrv = memory.getOldContextValue(addr+ActualQ[2])
                                print(addrv)
```

Cada que entramos a un ERA creamos un nuevo contexto para la memoria local y temporal,

```
    if ActualQ[1] == "ERA":
        memory.createLocalTemporal()
```

Una vez que terminamos con la ejecución de la función el contexto se elimina, con la función freeFunctionMemory()

```
    if ActualQ[1] == "END":
        memory.freeFunctionMemory()
        backup = []
        operationstack = []
```

Especificación gráfica de CADA estructura de datos usada para manejo de scopes (Memoria Local, global, etc..) ○ Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución).

Se usaron 5 estructuras relevantes durante el manejo de scopes en ejecución, son las siguientes:

La asociación entre las direcciones virtuales generadas en compilación, y las reales es manejada directamente desde el registro de memoria y los contextos que se generan en las funciones.

Arreglo BACKUP

('+', 1, 1, 'T1')	('PRINT, "Hello!!")	('=' 1 'local2')	...	('END', 'FIBO')

Este arreglo se usa como auxiliar de la fila de cuádruplos para poder inyectar operaciones al frente del quadruplo en caso de un do while

Arreglo OPERSTIONSTACK

('=', 1, 1, 'T1')	('PRINT, "Hello!!")	('=' 1 'local2')	...	('GOTOV', '23')	. . .	('END', 'FACTO')

Este arreglo se usa como auxiliar de las operaciones que se van a repetir en un do while, y es el que se inyecta al frente de la fila de cuádruplos hasta que se sale de la condición cíclica.

Arreglo FUNCSTACK

('INIT', 'FIBO')	('PRINT, "Hello!!")	('=' 1 'local2')	...	('INIT', 'FACTO')	. . .	('END', 'FACTO')

Funcstack es un arreglo que se usa como índice de funciones durante la ejecución, este se usa para inyectar la función a resolver al frente de la fila de cuádruplos en el momento que se llama a ella.

Arreglo MEMORYREG

('INT4', ADDR)	(T1, ADDR)	(T3,ADDR)	...	('T22, ADDR')

Memoryreg es un registro de los accesos de memoria para poder rápidamente conseguir la dirección de memoria de una variable en particular con solo saber su nombre

Y el arreglo de memoria demostrado en la sección anterior

```
def createMemory(self): #      G              L              T              C
    self.memoria =      [[{},{},{},{}], [[{},{},{},{}]],[[{},{},{},{}]],[{},{},{},{}]]
```

El arreglo de memoria está segmentado en 8 subtipos, dependiendo de su scope y su tipo; cada scope tiene sus 4 tipos que son integer, double, boolean y string.

Pruebas del funcionamiento del Lenguaje

Prueba #1: Fibonacci iterativo

```
lugstat lugstattest;
var input: int;

func fibo : int ( var limite: int; )
var local1, local2, cont, res: int;  {
  print("The first N fibo elements are...");
  local1 = 1;
  local2 = 1;
  cont =1;
  if(limite == 1){
    print("2");}
  else{
  print(local1+local2);
  res = local1+local2;
  do {
    local1 = local2;
    local2 = res;
    res = local1+local2;
    cont = cont+1;
    print(res);
    }while (cont <= limite-1);
    };}
{
print("Input N for fibo non recursive");
read(input);
fibo(input)
}
```

Output en Terminal:

Input N for fibo non recursive

(Input del usuario)
3

Output en Terminal:

The first N fibo elements are...
2
3
5

Cuadruplos:

('INIT', 'fibo'),
('PRINT', '"The first N fibo elements are..."'),
('=', 1, 'local1'),
('=', 1, 'local2'),

```
('=', 1, 'cont'),
('==', 1, 'limite', 't1', 25000),
(6, 'GOTOF', 't1', 9),
('PRINT', '"2"'), (8, 'GOTO', 0, 23),
('+', 'local2', 'local1', 't2', 20000),
('PRINT', 't2'),
('+', 'local2', 'local1', 't3', 20001),
('=', 't3', 'res'), ('DO', 13),
('=', 'local2', 'local1'),
('=', 'res', 'local2'),
('+', 'local2', 'local1', 't4', 20002),
('=', 't4', 'res'),
('+', 1, 'cont', 't5', 20003),
('=', 't5', 'cont'),
('PRINT', 'res'),
('-', 1, 'limite', 't6', 20004),
('<=', 't6', 'cont', 't7', 25001),
(23, 'GOTOV', 13),
(23, 'END', 'fibo'),
('PRINT', '"Input N for fibo non recursive"'),
('READ', 'input'),
(26, 'ERA', 'fibo'),
(27, 'PARAM', 'input', 'limite'),
(28, 'GOSUB', 'fibo', 1)]
```

Prueba #2 Fibonacci Recursivo

```
lugstat lugstattest;
func fiborec : int ( var l1, l2, cont, res, limit: int; )
{
if( limit == 1){
  print(2);};
if(cont == 1){
  print(2);};
if (limit >= 1){
  if (cont <= limit-1){
    l1 = l2;
    l2 = res;
    res = l1+l2;
    cont = cont+1;
    print(res);
    fiborec(l1, l2, cont, res, limit)};
};
```

Output en Terminal:

```
fibo recursive, first 4 elements
2
3
5
8
```

```
}
{
print("fibo recursive, first 4 elements");
fiborec(1,1,1,2, 4)
}
```

Cuadruplos:

deque([('INIT', 'fiborec'),
('==', 1, 'limit', 't1', 25000)
, (2, 'GOTOF', 't1', 4)
, ('PRINT', 2),
('==', 1, 'cont', 't2', 25001)
, (5, 'GOTOF', 't2', 7),
('PRINT', 2)
, ('>=', 1, 'limit', 't3', 25002),
(8, 'GOTOF', 't3', 26)
('-', 1, 'limit', 't4', 20000),
('<=', 't4', 'cont', 't5', 25003)
, (11, 'GOTOF', 't5', 26)
, ('=', 'l2', 'l1'),
('=', 'res', 'l2'),
('+', 'l2', 'l1', 't6', 20001),
('=', 't6', 'res'),
('+', 1, 'cont', 't7', 20002),
('=', 't7', 'cont'),
('PRINT', 'res'),
(20, 'ERA', 'fiborec'),
(21, 'PARAM', 'l1', 'l1'),
(22, 'PARAM', 'l2', 'l2'),
(23, 'PARAM', 'cont', 'cont'),
(24, 'PARAM', 'res', 'res'),
(25, 'PARAM', 'limit', 'limit'),
(26, 'GOSUB', 'fiborec', 1),
(26, 'END', 'fiborec'),
('PRINT', '"fibo recursive, first 4 elements"'),
(28, 'ERA', 'fiborec'),
(29, 'PARAM', 1, 'l1'),
(30, 'PARAM', 1, 'l2'),
(31, 'PARAM', 1, 'cont'),
(32, 'PARAM', 2, 'res')
, (33, 'PARAM', 4, 'limit')
, (34, 'GOSUB', 'fiborec', 1)]

Prueba #3: Factorial iterativo

```
lugstat lugstattest;
var input: int;
func facto : int (var input: int;)
var res, i : int;{
  res = 1;
  i = 2;
  do {
    res = res * i;
    i = i+1;
    }while(i <= input);
  print("The N Factorial is...");
  print(res);}
{
print("Input N for fact non recursive");
read(input);
facto(input)
}
```

Output en Terminal:

Input N for fact non recursive

(Input del usuario)
3

Output en Terminal:
The N Factorial is...
6

Cuadruplos:

```
[('INIT', 'facto'),
('=', 1, 'res'),
('=', 2, 'i'),
('DO', 3),
('*', 'i', 'res', 't1', 20000),
('=', 't1', 'res'),
('+', 1, 'i', 't2', 20001),
('=', 't2', 'i'),
('<=', 'input', 'i', 't3', 25000),
(9, 'GOTOV', 3),
('PRINT', '"The N Factorial is..."'),
('PRINT', 'res'),
(11, 'END', 'facto'),
('PRINT', '"Input N for fact non recursive"'),
('READ', 'input'),
(14, 'ERA', 'facto'),
(15, 'PARAM', 'input', 'input'),
(16, 'GOSUB', 'facto', 1)]
```

Prueba #4: Factorial recursivo

```
lugstat lugstattest;
var input: int;
func factorec : int (var res, i , lim: int;)
{
if (i <=lim){
res = res*i;
i = i+1;
factorec(res, i, lim)
if (i >lim){
print(res);
};};}

{
print("fact recursive for 4");
factorec(1, 2, 4)
}
```

Output en Terminal:

```
fact recursive for 4
24
```

Cuadruplos:

```
[('INIT', 'factorec'),
('<=', 'lim', 'i', 't1', 25000),
(2, 'GOTOF', 't1', 15)
, ('*', 'i', 'res', 't2', 20000),
('=', 't2', 'res'),
('+', 1, 'i', 't3', 20001),
('=', 't3', 'i'),
(8, 'ERA', 'factorec'),
(9, 'PARAM', 'res', 'res'),
(10, 'PARAM', 'i', 'i'),
(11, 'PARAM', 'lim', 'lim'),
(12, 'GOSUB', 'factorec', 1),
('>', 'lim', 'i', 't4', 25001),
(13, 'GOTOF', 't4', 15),
('PRINT', 'res'),
(15, 'END', 'factorec'),
('PRINT', '"fact recursive for 4"'),
(17, 'ERA', 'factorec'),
(18, 'PARAM', 1, 'res'),
(19, 'PARAM', 2, 'i'),
(20, 'PARAM', 4, 'lim'),
(21, 'GOSUB', 'factorec', 1)]
```

Listados perfectamente documentados del proyecto

Incluir comentarios de ***Documentación***, es decir: para cada módulo, una pequeña explicación de qué hace, qué parámetros recibe, qué genera como salida y cuáles son los módulos más importantes que hacen uso de él.

```
def FILL(elem1, elem2): # Esta funcion le agrega la linea de salida de un if para hacer el "salto". tambien es usado en el else.

    for i in range ( 0, Quad.qsize()):
        if Quad.queue[i][0] == elem1:
            # Encuentra la linea del salto que se encuentra al inicio del quadruplo
            a = Quad.queue[i][0]
            b = Quad.queue[i][1]
            c = Quad.queue[i][2]
            d = Quad.queue[i][3]
            #  Regenera el quadruplo y le reasigna el salto faltante al final.
            Quad.queue[i] = (a, b, c, elem2)
```

```
def typetostr(element): # Esta funcion recibe un <class 'type'> y lo convierte a el formato de nuestro cubo semantico que usa strings de los tipos.

        #Si ya esta en el formato apropiado solo regresa lo que le llego
        stringvalues = {'int', 'double', 'float', 'string', 'bool'}
        if element in stringvalues:
                return element

        else: # Si no, lo regresa en el formato apropiado.
                if element is int:
                        return 'int'
                if element is float:
                        return 'double'
                if element is str:
                        return 'string'
                if element is bool:
                        return 'bool'
```

```
    def freeFunctionMemory(self): # Esta función no recibe nada y libera el contexto actual
        self.memoria[1].pop() #Al terminar una función eliminamos las variables locales
        self.memoria[2].pop() #Al terminar una función eliminamos las variables Temporales
```

```
    def createLocalTemporal(self): # Esta función no recibe nada y genera un nuevo contexto.
        self.memoria[1].append([{},{},{},{}]) #Al crear una función siempre crear una nueva dirección
        self.memoria[2].append([{},{},{},{}]) #Al crear una función eliminamos las variables Temporales

    def getActualContextValue(self, direccion): # Est a función recibe una dirección y regresa el valor que tiene
        if abs(direccion) // 10000 == 0: #Guardamos valor en Globales
            return self.memoria[0][((abs(direccion) % 10000) // 2500)][(abs(direccion) % 10000) % 2500]
        elif abs(direccion) // 10000 == 1: #Guardamos valor en Locales
            return self.memoria[1][-1][((abs(direccion) % 10000) // 2500)][(abs(direccion) % 10000) % 2500]
        elif abs(direccion) // 10000 == 2:
            return self.memoria[2][-1][((abs(direccion) % 10000) // 2500)][(abs(direccion) % 10000) % 2500]
        elif abs(direccion) // 10000 == 3: #Guardamos valor en Constantes
            return self.memoria[3][((abs(direccion) % 10000) // 2500)][(abs(direccion) % 10000) % 2500]

        else:
            print("Address not found")
```

```
    def addv(self, fname, vname, vtype,loc):
# Esta función se usa para agregar las variables a un directorio de funciones
        if fname in self.listaf:
#fname es el nombre de la función.
            #print ("Function exists")
            access = self.listaf[fname]
#En caso de que si exista le asigna el objeto a access para meterse más profundamente al objeto
            if access['fvars'].search(vname) == True:
# Verifica si exista la variable ya
                print("Variable already exists")
            else:
                access['fvars'].add(vname, vtype,loc)
# Si no existe, agrega la variable.
        else:
            print("Function does not exist")
# Si fname no existe en el directorio, avisa que no existe.
```

```
def p_vars(p):
#Esta regla es muy importante para nuestro proyecto, ya que maneja todos los tipos de asignacion de memoria, registro y contaduria de parámetros y variables locales.
    '''
    vars : VAR vars1
    '''
    global vmcounter
    global vfcounter
    global pfcounter
    global pftypestack
```

```
    global pfboolstackcond
    global Li
    global Ld
    global Lb
    global Ls
    if p[-4] == "lugstat":
     #Significa que vengo del main por lo tanto agrego a mi funcion main;
        for i in range(len(FuncionActual)):
                if(TipoActual[0] == 'int'):
                        if len(ValorArreglo) > 0: # Caso para agregar arreglos
                            arreglo = {
                                'name' : FuncionActual[i],
                                'inicio' : Li,
                                'final'  : Li + ValorArreglo[0] - 1,
                                'type' : TipoActual[0]
                            }
                            DirectorioFunciones.addarreglo(p[-3],arreglo)
                            for j in range(ValorArreglo[0]):
                                memory.addMemoryValue(Li,70)
                                MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                                Li = Li + 1 # Se suma 1 a el contador de variables integer local una vez que se registre, todas
se inicializan con un valor de 70.
                        else:
                            DirectorioFunciones.addv(p[-3],FuncionActual[i],TipoActual[0],Li)
                            if(Li == 10000):
                                memory.addMemoryValue(Li,70)
                                MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                                Li = Li + 1 # Si no es un arreglo simplemente lo registra y se suma1 al registro de Li (caso
default)
                            else:
                                memory.addMemoryValue(Li,70)
                                MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                                Li = Li + 1 # Se registra la variable
                if(TipoActual[0] == 'double'): # Son los mismos casos para double . .
                        if len(ValorArreglo) > 0:
                            arreglo = {
                                'name' : FuncionActual[i],
                                'inicio' : Ld,
                                'final'  : Ld + ValorArreglo[0] - 1,
                                'type' : TipoActual[0]
                            }
                            DirectorioFunciones.addarreglo(p[-3],arreglo)
                            for j in range(ValorArreglo[0]):
                                memory.addMemoryValue(Ld,70)
                                MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
                                Ld = Ld + 1 # Arreglo de doubles
                        else:
```

```
                    DirectorioFunciones.addv(p[-3],FuncionActual[i],TipoActual[0],Ld) # Se agrega a la tabla de
variables
                    if(Ld == 12500):
                        memory.addMemoryValue(Li,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
                        Ld = Ld + 1 # Caso default
                    else:
                        memory.addMemoryValue(Li,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                        Ld = Ld + 1 #Caso Regular para registro de memoria
            if(TipoActual[0] == 'bool'): # Llega un booleano y se agrega a memoria y tabla de variables
                if len(ValorArreglo) > 0:
                    arreglo = {
                        'name' : FuncionActual[i],
                        'inicio' : Lb,
                        'final'  : Lb + ValorArreglo[0] - 1,
                        'type' : TipoActual[0]
                    }
                    DirectorioFunciones.addarreglo(p[-3],arreglo) # Generacion de arreglo de bools
                    for j in range(ValorArreglo[0]):
                        memory.addMemoryValue(Lb,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
                        Lb = Lb + 1 # Se suma al contador de Locales Bool
                else:
                    DirectorioFunciones.addv(p[-3],FuncionActual[i],TipoActual[0],Lb) # Se agrega a la tabla de
variables
                    if(Lb == 15000):
                        memory.addMemoryValue(Lb,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
                        Lb = Lb + 1
                    else:
                        memory.addMemoryValue(Lb,70)
                        Lb = Lb + 1
            if(TipoActual[0] == 'string'): # Mismos casos para string
                if len(ValorArreglo) > 0:
                    arreglo = {
                        'name' : FuncionActual[i],
                        'inicio' : Ls,
                        'final'  : Ls + ValorArreglo[0] - 1,
                        'type' : TipoActual[0]
                    }
                    DirectorioFunciones.addarreglo(p[-3],arreglo) # Generacion de arreglo
                    for j in range(ValorArreglo[0]):
                        memory.addMemoryValue(Ls,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                        Ls = Ls + 1  # Se suma 1 al contador
                else:
```

```
                    DirectorioFunciones.addv(p[-3],FuncionActual[i],TipoActual[0],Ls) # Se agrega a tabla de
variables
                    if(Ls == 12500):
                        memory.addMemoryValue(Ls,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                        Ls = Ls + 1
                    else:
                        memory.addMemoryValue(Ls,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                        Ls = Ls + 1 # Se agrega al registro de memoria, valor inicial y se suma 1 al contador de
Locales String
            vmcounter+=1
            #print(FuncionActual[i], "@#!#!@")
        Li = 10000
        Ld = 12500
        Lb = 15000 # RESET de contadores!
        Ls = 17500
        FuncionActual.clear()
        TipoActual.clear()
        ValorArreglo.clear()
    else:
        if p[-1] == '(': #Vengo desde FUNC soy parte de una funcion, Estos son los casos para parametros.
            #print(currentf, "$@#$@#")
            for i in range(len(FuncionActual)):
                if(TipoActual[0] == 'int'):
                        if len(ValorArreglo) > 0:
                            arreglo = {
                                'name' : FuncionActual[i],
                                'inicio' : Li,
                                'final'  : Li + ValorArreglo[0] - 1,
                                'type' : TipoActual[0]
                            }
                            DirectorioFunciones.addarreglo(p[-5],arreglo)
                            for j in range(ValorArreglo[0]):
                                memory.addMemoryValue(Li,70)
                                MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                                paramstack.append(FuncionActual[i])
                                # lo unico diferentes es que se tiene el stack de parametros para agregarlo al dirfunc
                                Li = Li + 1
                        else:
                            DirectorioFunciones.addv(p[-5],FuncionActual[i],TipoActual[0],Li)
                            memory.addMemoryValue(Li,70)
                            MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                            paramstack.append(FuncionActual[i])
                            Li = Li + 1
                if(TipoActual[0] == 'double'):
                        if len(ValorArreglo) > 0:
```

```
            arreglo = {
                'name' : FuncionActual[i],
                'inicio' : Ld,
                'final'  : Ld + ValorArreglo[0] - 1,
                'type' : TipoActual[0]
            }
            DirectorioFunciones.addarreglo(p[-5],arreglo)
            for j in range(ValorArreglo[0]):
                memory.addMemoryValue(Ld,70)
                MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
                paramstack.append(FuncionActual[i])
                Ld = Ld + 1
        else:
            DirectorioFunciones.addv(p[-5],FuncionActual[i],TipoActual[0],Ld)
            memory.addMemoryValue(Ld,70)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
            paramstack.append(FuncionActual[i])
            Ld = Ld + 1
    if(TipoActual[0] == 'bool'):
        if len(ValorArreglo) > 0:
            arreglo = {
                'name' : FuncionActual[i],
                'inicio' : Lb,
                'final'  : Lb + ValorArreglo[0] - 1,
                'type' : TipoActual[0]
            }
            DirectorioFunciones.addarreglo(p[-5],arreglo)
            for j in range(ValorArreglo[0]):
                memory.addMemoryValue(Lb,70)
                MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
                paramstack.append(FuncionActual[i])
                Lb = Lb + 1
        else:
            DirectorioFunciones.addv(p[-5],FuncionActual[i],TipoActual[0],Lb)
            memory.addMemoryValue(Lb,70)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
            paramstack.append(FuncionActual[i])
            Lb = Lb + 1
    if(TipoActual[0] == 'string'):
        if len(ValorArreglo) > 0:
            arreglo = {
                'name' : FuncionActual[i],
                'inicio' : Ls,
                'final'  : Ls + ValorArreglo[0] - 1,
                'type' : TipoActual[0]
            }
            DirectorioFunciones.addarreglo(p[-5],arreglo)
```

```
                    for j in range(ValorArreglo[0]):
                        memory.addMemoryValue(Ls,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                        paramstack.append(FuncionActual[i])
                        Ls = Ls + 1
                else:
                    DirectorioFunciones.addv(p[-5],FuncionActual[i],TipoActual[0],Ls)
                    memory.addMemoryValue(Ls,70)
                    MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                    paramstack.append(FuncionActual[i])
                    Ls = Ls + 1
            pfcounter+=1
            #print(TipoActual[0], " of type")
            #print(p[-1], "fds")
            pftypestack.append(TipoActual[0])
            #print(pfcounter, "params!")
        FuncionActual.clear()
        TipoActual.clear()

if p[-1] == ";":
 #N linea de Variables (Usualmente de otro tipo).
    for i in range(len(FuncionActual)):
            if(TipoActual[0] == 'int'):
                if len(ValorArreglo) > 0:
                    arreglo = {
                        'name' : FuncionActual[i],
                        'inicio' : Li,
                        'final'  : Li + ValorArreglo[0] - 1,
                        'type' : TipoActual[0]
                    }
                    DirectorioFunciones.addarreglo(currentf[-1],arreglo)
                    for j in range(ValorArreglo[0]):
                        memory.addMemoryValue(Li,70)
                        MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                        Li = Li + 1
                else:
                    DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Li)
                    memory.addMemoryValue(Li,70)
                    MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
                    Li = Li + 1
            if(TipoActual[0] == 'double'):
                if len(ValorArreglo) > 0:
                    arreglo = {
                        'name' : FuncionActual[i],
                        'inicio' : Ld,
                        'final'  : Ld + ValorArreglo[0] - 1,
                        'type' : TipoActual[0]
```

```
            }
            DirectorioFunciones.addarreglo(currentf[-1],arreglo)
            for j in range(ValorArreglo[0]):
                memory.addMemoryValue(Ld,70)
                MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
                Ld = Ld + 1
        else:
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Ld)
            memory.addMemoryValue(Ld,70)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
            Ld = Ld + 1
    if(TipoActual[0] == 'bool'):
        if len(ValorArreglo) > 0:
            arreglo = {
                'name' : FuncionActual[i],
                'inicio' : Lb,
                'final'  : Lb + ValorArreglo[0] - 1,
                'type' : TipoActual[0]
            }
            DirectorioFunciones.addarreglo(currentf[-1],arreglo)
            for j in range(ValorArreglo[0]):
                memory.addMemoryValue(Lb,70)
                MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
                Lb = Lb + 1
        else:
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Lb)
            memory.addMemoryValue(Lb,70)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
            Lb = Lb + 1
    if(TipoActual[0] == 'string'):
        if len(ValorArreglo) > 0:
            arreglo = {
                'name' : FuncionActual[i],
                'inicio' : Ls,
                'final'  : Ls + ValorArreglo[0] - 1,
                'type' : TipoActual[0]
            }
            DirectorioFunciones.addarreglo(currentf[-1],arreglo)
            for j in range(ValorArreglo[0]):
                memory.addMemoryValue(Ls,70)
                MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
                Ls = Ls + 1
        else:
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Ls)
            memory.addMemoryValue(Ls,70)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
            Ls = Ls + 1
```

```
            if currentf[-1] != currentf[0] and pfboolstackcond == True:
               # Aqui se agrega al stack de tipos de parametros que se usan para el chequeo de tipos en las
llamadas a funciones
                pftypestack.append(TipoActual[0])
                pfcounter+=1
                #Tambien se cuenta el numero de parametros desde aqui, gracias a los booleanos. Currentf[0] es
el main, y nunca tiene parametros.
            if currentf[-1] != currentf[0] and pfboolstackcond == False:
                        # vfcounter revisa la cantidad de variables locales tiene una funcion, que se encuentran en
una linea entre despues de )
                vfcounter+=1
            #print("STATUS:", currentf)
    FuncionActual.clear()
    TipoActual.clear()
    ValorArreglo.clear()
  if p[-1] == ')': # Variables locales de una FUNC, osea justo despues de la declaracion basica de la funcion.
    for i in range(len(FuncionActual)):
        if(TipoActual[0] == 'int'):
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Li)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Li, 70))
            #memory.addMemoryValue(Li,70)
            # Se agregan los integers a la funcion actual que es currentf[-1] y se hace el registro de memoria
            Li = Li + 1
        if(TipoActual[0] == 'double'):
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Ld)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Ld, 70))
            #memory.addMemoryValue(Ld,70)
            # Se agregan los doubles a la funcion actual que es currentf[-1] y se hace el registro de memoria
            Ld = Ld + 1
        if(TipoActual[0] == 'bool'):
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Lb)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Lb, 70))
            #memory.addMemoryValue(Lb,70)
            # Se agregan los bools a la funcion actual que es currentf[-1] y se hace el registro de memoria
            Lb = Lb + 1
        if(TipoActual[0] == 'string'):
            DirectorioFunciones.addv(currentf[-1],FuncionActual[i],TipoActual[0],Ls)
            MemoryREG.append((FuncionActual[i],TipoActual[0], Ls, 70))
            #memory.addMemoryValue(Ls,70)
            # Se agregan los strings a la funcion actual que es currentf[-1] y se hace el registro de memoria
            Ls = Ls + 1
        vfcounter+=1
        # Se suma al contador de variables locales que tiene una funcion que despues se agrega al directorio
de funciones.
    Li = 10000
    Ld = 12500
    Lb = 15000 # RESET DE CONTADORES
```

```
Ls = 17500
FuncionActual.clear()
TipoActual.clear()
ValorArreglo.clear()
```

Manual de usuario

Pre requisitos:

Dependencias necesarias para versión funcional del código

- Numpy que usamos como np
- Pandas que usamos como pd
- Matplotlib que usamos como plt
- Sklearn.datasets
- Sklearn clusters
- Scipy.stats
- Seabron que usamos como sb
- Sympy

La instalación de estas dependencias se hacen de la siguiente manera

pip/ppip3 install nombre_dependencia desde consola.

Es necesario tener una versión de python 3.0 o mayor instalada.

Ejecución

Es necesario contar con el archivo inputf.txt, aquí es donde tendremos nuestro código, para la lectura de un input diferente es necesario reemplazar el nombre del archivo por el que se desea compilar dentro de LUGSTAT.py en dondé en la linea 117 encontraremos InputF

```
#Aquí se ingresa el archivo a compilar

InputF= open("inputf.txt", "r")
```

Para ejecutar el programa solo es necesario el comando pyhton3/py LUGSTAT.py

Tipos de variables

- **Int**
- **Double**
- **Bool**
- **String**

Manejo de Sintaxis:

Main
 	
LUGSTAT es el token de inicio para lugstat.
ID es el nombre de el main
SCOLON es ;
Lugstat2 es la función que va a las variables
Lugstat3 es la función que va a los módulos
	`lugstat : LUGSTAT ID SCOLON lugstat2 lugstat3 block`
	
	`lugstat mimain;`
	`var mivar1 : int;`
	`func mifunc : int (param1: int;) {}`
	`{}`

Variables
 	
“vars1” es la gramática para definir variables, brinca desde la regla vars donde esta el token “var” asign2 llama a la regla de declaracion de arreglos y matrices.
 	
LUGSTAT solo permite declaraciones genéricas. Estas declaraciones se pueden juntar en una sola con commas si son del mismo tipo.
	`vars : VAR vars1`
	
	`vars1 : ID COMMA vars1`
	`| ID COLON tipo SCOLON lugstat2`
	`| ID asign2 COLON tipo SCOLON`
	`| ID asign2 COMMA vars1`
	
	`var mivar1 : int;`
	`var mibool1, mibool2 :bool;`
	`var midouble1, midobule2 :double;`

Funciones
 	
Modules es la declaración de los módulos o funciones adicionales. Inicia con FUNC
ID es el nombre del módulo
Tipo es el tipo de retorno
 	
modules2 manda a llamar la declaración de variables, primero para parámetros luego las variables locales de la función.
 	
Funblock es el bloque de la función.
	`modules : FUNC ID COLON tipo OPAREN modules2 CPAREN modules2 funblock`
	
	`func mifunc : int (param1: int;)`
	`var mivlocal1: int;`
	`{}`

Bloques
 	
Los bloques son estatutos rodeados por brackets.
 	
Dentro de los estatutos están todas las funciones y operaciones.
	`block : OBRACKET block2 CBRACKET`
	
	`'''`
	`block2 : estatuto`
	`| estatuto block2`
	`| empty'''`

Estatutos basicos
 	
Estos son los estatutos básicos, cada uno manda
	`'''`
	`estatuto : asign`
	`| cond`
	`| escrt`
	`| plot`
	`| count`

a su regla.
Entre ellos están el de asignación, las condiciones, estructura, do while, llamada a función y al read line.
	`| countif`
	`| metodos`
	`| dwhile`
	`| readln`
	`| funccall`
	`'''`

Asignacion
--- 	
LUGSTAT permite varios tipos de asignación, ya sea el resultado de una expresión, igualar variables, igualar el contenido de un arreglo y operaciones de arreglos en general.
	`'''`
	`asign : ID EQUALS expresion SCOLON`
	`| ID EQUALS ID SCOLON`
	`| ID EQUALS ID asign2 SCOLON`
	`| ID asign2 EQUALS ID SCOLON`
	`| ID assign2 EQUALS expresion SCOLON`
	`| ID asign2 EQUALS ID asign2 SCOLON`
	`'''`
	
	`Var1 = 1+1*(100/10);`
	`Var1 = Var2;`
	`Var1 = Arr1[1];`
	`Arr1[2+1] = Arr2[3+i];`
	`. . .`
	`Arr1[2] = 1*3;`

Escritura
--- 	
El estatuto de escritura de LUGSTAT escrt permite juntar predicados con commas y imprimirlos, ya sean variables, expresiones o strings.
	`'''escrt : PRINT OPAREN ID escrt2 CPAREN SCOLON`
	`| PRINT OPAREN expresion CPAREN SCOLON`
	`| PRINT OPAREN STRING CPAREN SCOLON`
	`'''`
	
	`Print (var1);`
	`print(var1, var2);`
	`print(1+1);`
	`print("hola mundo!");`

Condiciones
--- 	
LUGSTAT soporta estatutos condicionales tipo IF y IF - ELSE para expresiones con operadores relacionales == , < ,> ,<= ,>= ,!=
	`'''cond : IF OPAREN expresion CPAREN ifblock SCOLON`
	`| IF OPAREN expresion CPAREN ifblock ELSE ifblock SCOLON`
	`'''`
	
	`If (a > b) {};`
	`. . .`
	`If (a > b) {} else {};`

Expresiones
--- 	
El manejo de expresiones de LUGSTAT está basado en el de littleduck2019.
	`'''expresion : exp`
	`| expresion RELOP exp`
	`'''`
	
	`'''exp : termino`
	`| termino PLUS exp`
	`| termino MINUS exp`
	`'''`
	
	`'''termino : factor`
	`| factor MULT termino`
	`| factor DIV termino`

	'''
	
	'''factor : OPAREN expresion CPAREN
	  | varcte
	'''
	
	'''varcte : ID
	  | ID asign2
	  | NUMERIC
	  | NUMBER
	'''

Do-While La manera que hace lugstat sus ciclos es usando DO - WHILE	'''dwhile : DO wblock WHILE OPAREN dwhileconds CPAREN SCOLON '''
Read Line LUGSTAT soporta inputs por el usuario usando la función read, lee y asigna a la variable lo que se leyó.	''' readln : READ OPAREN ID CPAREN SCOLON ''' read(k);

Funciones especiales

mean(Arreglo de enteros);

Función que imprime el promedio de una lista, recibe como parámetro un arreglo de enteros.

median(Arreglo de enteros);

Función que imprime la mediana de una lista, recibe como parámetro un arreglo de enteros.

mode(Arreglo de enteros);

Función que imprime la moda de una lista, recibe como parámetro un arreglo de enteros.

stdv(Arreglo de enteros);

Función que regresa la mediana de una lista, recibe como parámetro un arreglo de enteros.

kmeans(Nclusters,Arreglo de enteros);

Función que genera una gráfica a partir del número de clusters, recibe como parámetro un entero y un arreglo de enteros.

derl(i1,i2,i3);

Función que genera una gráfica a partir de los parámetros con distribución de erlang, recibe como parámetro tres entero y un arreglo de enteros.

dbern(Nclusters,Arreglo de enteros);

Función que genera una gráfica con distribución de bernoulli a partir del tamaño y esperanza, recibe como parámetro un entero y un doble.

dpoi(Nclusters,Arreglo de enteros);

Función que genera una gráfica con distribución poisson a partir del tamaño y esperanza, recibe como parámetro un entero y un doble.

transpose(Matriz de enteros);

Función que despliega una matriz en su forma transpuesta recibe como parámetro una matriz de enteros o dobles.

inverse(Matriz de enteros);

Función que despliega una matriz en su forma inversa recibe como parámetro una matriz de enteros o dobles.

rotate(Matriz de enteros);

Función que despliega una matriz en su forma rotada recibe como parámetro una matriz de enteros o dobles.

ref(Matriz de enteros);

Función que despliega una matriz en su forma reducida recibe como parámetro una matriz de enteros o dobles.

rref(Matriz de enteros);

Función que despliega una matriz en su forma reducida escalonada recibe como parámetro una matriz de enteros o dobles.