

# DOCUMENTACIÓN FINAL



COMPIMOI

MOISÉS LÓPEZ NÚÑEZ

A01235387

5/Junio/2023



COMPIMOI	1
MOISÉS LÓPEZ NÚÑEZ	1
A01235387	1
<b>Descripción del Proyecto</b>	<b>3</b>
Propósito y Alcance:	3
Análisis de Requerimientos y descripción de los principales Test Cases.	3
Descripción del proceso de desarrollo	4
<b>Descripción del lenguaje</b>	<b>6</b>
Nombre del lenguaje:	6
Descripción genérica de las principales características del lenguaje	6
Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución.	6
<b>Descripción del compilador.</b>	<b>6</b>
Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.	6
Descripción del Análisis de Léxico	7
Descripción del Análisis de Sintaxis.	8
Descripción de Generación de Código Intermedio y Análisis Semántico.	11
Memoria de compilación, a continuación se define los rangos de los tipos de memoria de ejecución que se maneja:	12
Descripción de puntos neurálgicos:	23
Manejo de tipos, Cubo Semántico:	27
Descripción detallada del proceso de Administración de Memoria usado en la compilación	28
<b>DESCRIPCIÓN DE LA MÁQUINA VIRTUAL</b>	<b>33</b>
Equipo de cómputo, lenguaje y utilerías especiales usadas	33
Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura),	33
<b>PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :</b>	<b>35</b>
Fibonacci recursivo y cíclico	35
Código ejecutado	35
Cuádruplos generados:	36
Resultado	38
Factorial Cíclico y Recursivo	38
Código ejecutado	38
Cuádruplos generados:	39
Resultado	40
Find:	40
Código ejecutado	40
Cuádruplos generados:	41
Resultado:	43
Matrix Multiplication	43
Código ejecutado:	43

Cuádruplos generados:	45
Resultado	50
Vector Manipulation	51
Cuádruplos generados:	51
Resultado	53
Bubble Sort	54
Código Ejecutado:	54
Cuádruplos generados	55
Resultado:	57
<b>DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO</b>	<b>57</b>
<b>Quick Reference</b>	<b>58</b>
Instalación:	58
VIDEODEMO	60

## Descripción del Proyecto

### Propósito y Alcance:

El propósito de este proyecto es aplicar todos los conocimientos aprendidos en la carrera de Ingeniería en Tecnologías Computacionales y aplicarlos en crear un compilador y máquina virtual que permite ejecutar código escrito siguiendo reglas que se definieron previamente. El alcance del proyecto es que nuestro compilador y máquina virtual ejecute funciones primordiales de programación que se definen más adelante en el documento. Se referirá el proyecto como CompiMoi, nombre dado por la combinación de Compilador y Moi que es nombre del autor.

### Análisis de Requerimientos y descripción de los principales Test Cases.

A continuación se enumerarán los requerimientos que se plantearon para este proyecto.

- CompiMoi debe leer código escrito por el programador y regresar código intermedio (cuádruplos) que será leído por una máquina virtual que ejecutará este conjunto de instrucciones.
- La máquina virtual debe de recibir código intermedio (cuádruplos) y ejecutar correctamente el conjunto de instrucciones generado.
- CompiMoi debe de generar memoria de compilación, este se encargará de guardar toda la información necesaria para compilar el código.
- La máquina virtual debe de administrar la memoria de ejecución, esta se encargará de administrar los valores de direcciones de variables dependiendo de su scope, así como también checar que no se rebase la memoria máxima establecida para todo la máquina virtual.

- CompiMoi y la máquina virtual deben de indicar errores tanto de ejecución como de compilación
- CompiMoi y la máquina virtual deben ser capaces de aceptar variables Int, Float, Boolean, String, tanto como Arrays de N dimensiones de estos mismos tipos
- CompiMoi y la máquina virtual deben de aceptar estatutos no lineales primordiales: While, If, IfElse.
- CompiMoi y la máquina virtual deben de aceptar declaración y llamadas a módulos creados.
- Los módulos pueden o no regresar un tipo de retorno.
- CompiMoi y la máquina virtual deben de poder realizar un find y un sort para Arrays de una dimensión.
- CompiMoi y la máquina virtual deben poder aceptar Fibonacci y Factorial en sus versiones recursivas y cíclicas.
- CompiMoi y la máquina virtual deben de aceptar multiplicación de matrices del mismo tamaño
- CompiMoi y la máquina virtual deben de aceptar leer y escribir datos de la terminal.

## Descripción del proceso de desarrollo

Para este proyecto se trabajaron avances con un alcance definido por el responsable de la materia, cabe recalcar que el autor en particular no siguió al cien por ciento estos avances, resultando en desfases del proyecto.

Los avances definidos por el educador fueron

Semana	Descripción Corta	Descripción
1	Análisis Léxico y Semántico	Durante este avance se empezó el análisis léxico, tomando en cuenta las palabras reservadas de mi lenguaje y sus respectivos tokens. Como nota cabe recalcar que en un futuro se fueron agregando a estas listas ya que no se definieron todas las requeridas para el proyecto
2	Semántica básica. Directorio de funciones Cubo Semántico	Se empezó la semántica básica, y diseño del directorio de funciones donde se guardará la información necesaria para el momento de compilación Cubo semántico necesario para poder obtener el tipo esperado después de la valoración de una expresión
3	Generación de código de expresiones	Inicio de generación de cuádruplos para expresiones aritméticas, respetando PEMDAS y generación de código para estatutos lineales
4	Generación de código de estatutos no lineales	Inicio de generación de cuádruplos para estatutos no lineales tales como If cases, Whiles.
5	Generación de	Inicio de generación de código para tomar en cuenta

	código de funciones	contextos de funciones
6	Máquina virtual y memoria de ejecución	Research de máquina virtual y diseño para una solucionar cambios de contexto a la hora de ejecutar los cuádruplos generados de funciones. Ejecución de expresiones sin cambios de contexto ni estatutos no lineales
7	Generación de código de Arrays y Ejecución de estatutos no lineales	Generación de código para arrays, tanto inicialización como acceso. Ejecución de estatutos no lineales en máquina virtual con cambios de contexto
8	1er avance documentación y propias funciones	Empezar documentación y generar semántica, léxico y ejecución de propias funciones definidas por el usuario para el compilador
9	Entrega final	Terminar documentación, implementar arrays en ejecución y debugear posibles casos extremos

Cabe recalcar que en algunos avances se tuvo que regresar a arreglar posibles errores encontrados a la hora de avanzar que se generaron en entregas anteriores

### Commits hechos a repositorio:

Python

```

29d36a6 - Moises Lopez : Bublle sort example 2023-06-05
2d60ce0 - Moises Lopez : updating readme 2023-06-05
0a8dac8 - Moises Lopez : adding readme changes and imporintg 2023-06-05
8f59e54 - Moises Lopez : extra cases, printing of matrices 2023-06-05
a5d45fc - Moises Lopez : reordering files all working no tests 2023-06-05
5fe0069 - Moises Lopez : boolean logic 2023-06-04
f408db1 - Moises Lopez : array accesing working 2023-06-04
cb32534 - Moises Lopez : updating readme 2023-06-03
15523ce - Moises Lopez : array accessing wip 2023-06-03
b8640ff - Moises Lopez : recursion, while, fibonacci, factorial function
calling working 2023-06-03
41a7da0 - Moises Lopez : virtual machine working, while ifs asignation
expresions 2023-06-02
ed532a2 - Moises Lopez : execution memory and virtual memory improvements,
quads con addreses 2023-06-02
332bd9d - Moises Lopez : cleaning code optmizing 2023-05-31
8d6e6a8 - Moises Lopez : array declaracion working 2023-05-31
a8fdd7f - Moises Lopez : adding final virtual memory and function calling.
Quads working right now 2023-05-30
7067bbc - Moises Lopez : updating readme 2023-05-29
d90af5b - Moises Lopez : updating readme 2023-05-29

```

```

94fe454 - Moises Lopez : virtual memory v1.0 working 2023-05-29
3c93a72 - Moises Lopez : adding size of functions to dir funciones 2023-05-29
244c6b6 - Moises Lopez : while working 2023-05-29
7e0daa9 - Moises Lopez : removing some logs 2023-05-28
c2a1ac5 - Moises Lopez : asignacion cuadruplos 2023-05-28
29f7575 - Moises Lopez : condiciones working 2023-05-27
299e780 - Moises Lopez : expression correct fondo falso 2023-05-26
6874cb4 - moises-lopez : wip 2023-05-23
ada9b60 - moises-lopez : dir function working 2023-05-23
a4f7413 - moises-lopez : cuarto avance semantica modulos y generacion de
codigo 2023-05-22
7e9ba91 - moises-lopez : dir funciones correcto 2023-05-22
cd29a27 - moises-lopez : tercer avance dir de funciones optimización y
semántica final 2023-05-14
fcfe4df - moises-lopez : segundo avance cuadruplos expresiones, cubo
semantico y utils 2023-05-08
da31c7d - moises-lopez : avance 1 2023-04-16
53811bc - Moises Lopez : Initial commit 2023-04-16

```

### Enseñanza:

Este proyecto sin lugar a dudas fue el más retador de mi carrera, me hizo aplicar todo lo aprendido durante mis 5 años en la institución. El conocimiento necesario para completar un proyecto de esta magnitud debe de ser muy grande. Aunque el responsable de la materia nos haya dado muchos sets de instrucciones para resolver la mayoría de los retos planteados en este proyecto, la capacidad de conocimiento necesario para implementarlos es demasiado grande.

Esta es mi segunda vez cursando esta materia, si alguien que está leyendo esto está apenas empezando la materia recomiendo ampliamente empezar desde el día 0 la realización de este proyecto, consume mucho tiempo, energía tanto mental como física. Estoy orgulloso de completar este proyecto con mi alcance planteado, soy sincero en decir que mi compilador no es perfecto pero conozco sus debilidades, lo que me hace capaz de tener la capacidad de arreglarlas en un futuro. Estoy consciente que no se puede generar un proyecto perfecto pero minimizar el total de errores.



## Descripción del lenguaje

### Nombre del lenguaje:

El proyecto se llama CompiMoi, es una mezcla entre compilador y el nombre del autor.

### Descripción genérica de las principales características del lenguaje

CompiMoi es un lenguaje de programación procedural, lo que significa que se lee de forma continua y el programador decide qué debe de hacer y cómo hacerlo. Todo esto para resolver algún caso específico definido por el programador.

Empieza con el nombre del programa, seguido de la definición de variables globales, no se aceptan expresiones en el scope global. Después se pueden declarar módulos, funciones reutilizables que reciben parámetros y que pueden regresar o no el valor de su algoritmo.

Le sigue la función principal Main, Main es lo primero que se ejecuta al ejecutar el programa, dentro de este puede haber llamadas a módulos, declaración de variables, estatutos no lineales, lectura de terminal, escritura en terminal. Todo lo posible que se puede hacer con el compilador se detallará más adelante.

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

Error	Descripción
Sintáctico	No existan palabras, variables declaradas etc.
Semántica	Error de tipos, type mismatch
Ejecución	Out of bounce, stack overflow, recursión infinita

### Descripción del compilador.

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

Este lenguaje de programación se realizó en un ambiente Windows y MacOS a la vez, utilizando el IDE Pycharm tanto para escribir el compilador tanto como para ejecutarlo. El lenguaje donde se realizó fue python, gracias a su facilidad de entendimiento tanto como su gran repertorio de librerías, se utilizó PLY (<https://www.dabeaz.com/ply/ply.html>) una librería para generar léxica y semántica con código de python. Como manejador de versiones se utilizó Git.

## Descripción del Análisis de Léxico

### Tokens utilizados:

Token Name	String o Regex
'INT_CTE'	r'\d+'
'FLOAT_CTE'	r'\d+\.\d+'
'BOOLEAN_CTE'	r'\b(True False)\b'
'STRING_CTE'	r'"([^\\n] (\\.))*?"'
'ID'	r'[a-zA-Z_][a-zA-Z_0-9]*'
'PLUS'	r'\+'
'MINUS'	r'\-'
'DIVIDE'	r'\/'
'MULTIPLY'	r'\*'
'EQUALS'	r'\= ='
'ASIGNATION'	r'\='
'COMMA'	r'\,'
'SEMICOLON'	r'\;'
'LEFTPARENTHESES'	r'\('
'RIGHTPARENTHESES'	r'\)'
'LEFTBRACE'	r'\{'
'RIGHTBRACE'	r'\}'
'LEFTBRACKET'	r'\['
'RIGHTBRACKET'	r'\]'
'LESSTHAN'	r'\<'



'LESSTHANOREQUALS'	r'\<\'='
'GREATERTHAN'	r'\>\'
'NOTEQUALS'	r'\!\'='
'THREEDOTS'	r'\.\.\.'
'RETURN'	r'return'
'AT'	r'\@\'
'PROGRAMA'	'Programa'
'VOID'	'void'
'MAIN'	'Main'
'FUNCTION'	'Function'
'VAR'	'var'
'IF'	'if'
'ELSE'	'else'
'WHILE'	'while'
'INT'	'int'
'FLOAT'	'float'
'BOOLEAN'	'boolean'
'READ'	'read'
'PRINT'	'print'
'NOTEQUALS'	r'\!\'='
'DOT'	r'\.'
'SQUARE'	'square'

## Descripción del Análisis de Sintaxis.

La sintaxis definida sigue el estándar de tokens y palabras reservadas en mayúsculas y reglas de sintaxis definidas extras en minúsculas

Python

```

calc : PROGRAMA ID SEMICOLON vars modulesaux functionmain

functionmain : MAIN LEFTPARENTHESSES RIGHTPARENTHESSES bloque

vars : VAR tipo varsAuxDeclaration SEMICOLON vars
      | empty

varsaux : COMMA ID varsaux
      |

tipo : INT
      | FLOAT
      | BOOLEAN

varsAuxDeclaration : idOrArrayDeclaration COMMA varsAuxDeclaration
      | idOrArrayDeclaration

idOrArrayDeclaration : ID
      | ID LEFTBRACKET arrayDimesionAux RIGHTBRACKET

arrayDimesionAux : INT_CTE INT_CTE
      | INT_CTE INT_CTE COMMA arrayDimesionAux

modulesaux : function modulesaux
      |

function : FUNCTION returnfunctionaux ID params bloque

returnfunctionaux : tipo
      | VOID

params : LEFTPARENTHESSES paramsaux RIGHTPARENTHESSES

paramsaux : tipo ID paramsaux
      | COMMA paramsaux
      | empty

bloque : LEFTBRACE vars bloqueaux returnaux RIGHTBRACE

returnaux : RETURN expresion SEMICOLON
      |

bloqueaux : estatuto bloqueaux
      |

estatuto : asignacion
      | condicion
      | escritura

```

```

| while
| functionCall SEMICOLON
| lectura
| specialArrayExpresion

```

**lectura** : READ LEFTPARENTHESES ID seen\_lectura RIGHTPARENTHESES SEMICOLON

**functionCall** : AT ID LEFTPARENTHESES paramsFunctionCall RIGHTPARENTHESES

**paramsFunctionCall** : expresion COMMA paramsFunctionCall  
| expresion

**while** : WHILE LEFTPARENTHESES expresion RIGHTPARENTHESES bloque SEMICOLON

**asignacion** : varcte ASIGNATION expresion SEMICOLON

**condicion** : IF LEFTPARENTHESES expresion RIGHTPARENTHESES bloque condicionaux

**condicionaux** : ELSE bloque SEMICOLON  
| SEMICOLON

**escritura** : PRINT LEFTPARENTHESES escrituraaux RIGHTPARENTHESES SEMICOLON

**escrituraaux** : escrituraaux2  
| escrituraaux2 COMMA escrituraaux

**escrituraaux2** : expresion  
| STRING\_CTE

**expresion** : exp  
| exp expresionaux exp

**expresionaux** : GREATERTHAN  
| LESSTHAN  
| LESSTHANOREQUALS  
| NOTEQUALS  
| EQUALS  
|

**exp** : termino  
| termino expaux

**expaux** : PLUS exp  
| MINUS exp

**termino** : factor

```

        | factor terminoaux

terminoaux : DIVIDE termino
            | MULTIPLY termino

factor : factoraux
        | varcte

factoraux : LEFTPARENTHESSES expresion RIGHTPARENTHESSES

varcte : ID
        | INT_CTE
        | FLOAT_CTE
        | functionCall
        | arrayAccesing
        | BOOLEAN_CTE

arrayAccesing : ID LEFTBRACKET arrayAccesingExpresionAux RIGHTBRACKET

arrayAccesingExpresionAux : expresion
                            | expresion COMMA arrayAccesingExpresionAux

```

## Descripción de Generación de Código Intermedio y Análisis Semántico.

CompiMoi genera código intermedio representado por cuádruplos, estos pueden tener 1 o 4 elementos.

El código intermedio es representado por un Array de Arrays, siendo cada elemento del Array principal un cuádruplo, que a su vez es un array por sí mismo, siguiendo la siguiente estructura

```

Python
[OPERATOR, LEFTOPERAND, RIGHTOPERAND, RESULT]

#Ejemplo de código intermedio:
Program Test;
Main(){
    A = 1 + 2;
    print(A);
}

```

```
};
#Resultado =
[
  []
  ['GOTO', '', '', 2]
  ['+', 1, 2, 't1']
  ['=', t1, '', A]
  ['PRINT', '', '', A]
]

#Resultado Direcciones:
[
  []
  ['GOTO', '', '', 2]
  ['+', 3001, 3002, 2001]
  ['=', 2001, '', 1001]
  ['PRINT', '', '', 1001]
]
```

Como nota: nuestro primer elemento del array es un array vacío, esto para empezar nuestro contador en el valor: 1

En el ejemplo, lo representado sería un programa sin módulos con un Main muy simple, también se observa la representación de sus cuádruplos tanto en su formato de nombres de variables como en la de direcciones. Los operadores mantienen su formato de string.

## Memoria de compilación, a continuación se define los rangos de los tipos de memoria de ejecución que se maneja:

```
Python
START_GLOBAL_VARS = 0
START_GLOBAL_VARS_INT = 0
END_GLOBAL_VARS_INT = 250
START_GLOBAL_VARS_FLOAT = 251
END_GLOBAL_VARS_FLOAT = 500
START_GLOBAL_VARS_BOOLEAN = 501
END_GLOBAL_VARS_BOOLEAN = 750
START_GLOBAL_VARS_CHAR = 751
END_GLOBAL_VARS_CHAR = 1000
END_GLOBAL_VARS = 1000

START_LOCAL_VARS = 1001
START_LOCAL_VARS_INT = 1001
END_LOCAL_VARS_INT = 1250
```

```

START_LOCAL_VARS_FLOAT = 1251
END_LOCAL_VARS_FLOAT = 1500
START_LOCAL_VARS_BOOLEAN = 1501
END_LOCAL_VARS_BOOLEAN = 1750
START_LOCAL_VARS_CHAR = 1751
END_LOCAL_VARS_CHAR = 2000
END_LOCAL_VARS = 2000

START_TEMPORAL_VARS = 2001
START_TEMPORAL_VARS_INT = 2001
END_TEMPORAL_VARS_INT = 2250
START_TEMPORAL_VARS_FLOAT = 2251
END_TEMPORAL_VARS_FLOAT = 2500
START_TEMPORAL_VARS_BOOLEAN = 2501
END_TEMPORAL_VARS_BOOLEAN = 2750
START_TEMPORAL_VARS_CHAR = 2751
END_TEMPORAL_VARS_CHAR = 3000
END_TEMPORAL_VARS = 3000

START_CONSTANTS = 3001
END_CONSTANTS = 4000

```

Se observa que las variables constantes empiezan en la dirección 3001 y terminan en 4000, y así sucesivamente para todas las demás.

Operador	Descripción
GOTO, “, “, result	Salto incondicional hacia el cuádruplo con el número que contenga la casilla de resultado del cuádruplo
Operadores aritméticos, lógicos y de comparación +, -, *, /, ==, !=, <, >, <=, >=	Se comportan de la manera convencional, tomando el left operando y el right operando para hacer la operación correspondiente y depositan su valor en el address que está en result
GOTOIF, condition, “, result	Salto condicionado a falso, toma como condición el left operand para hacer un salto al valor del result
PRINT, “, “, result	Escribe en terminal el valor del resultado
ENDFUNC, “, “, “	Indica el punto donde una función termina, libera la memoria de esta función y regresa el instruction pointer al valor guardado
GOSUB, functionName, “, address	Salto a la address de la función llamada

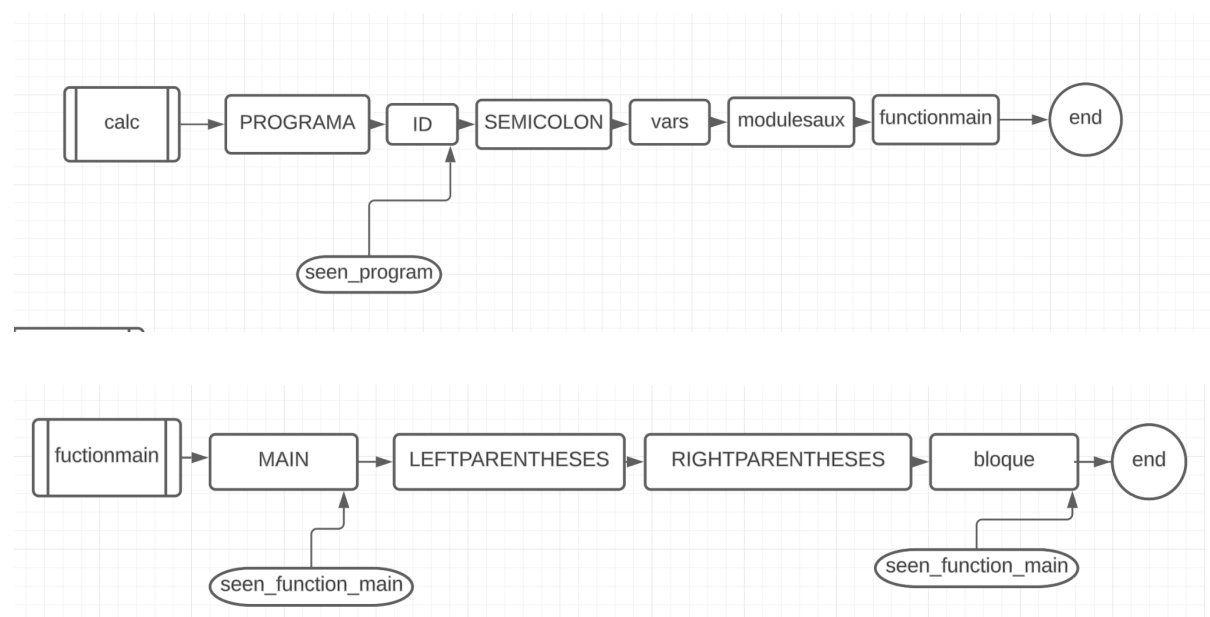
ERA, “, “, functionName	Prepara en memoria de ejecución la memoria requerida para la función llamada
PARAMETER, “, value, ParamNumber	Asigna el valor obtenido hacia el número de parámetro de la función previamente llamada
END, “, “, “	Indica el final del programa.
VERIFY, value, upperLimit, lowerLimit	Verifica que el valor esté dentro del rango de límites

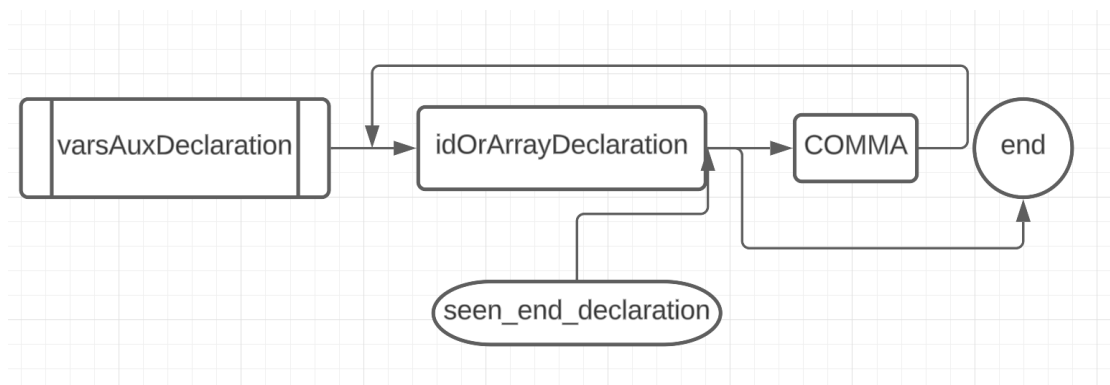
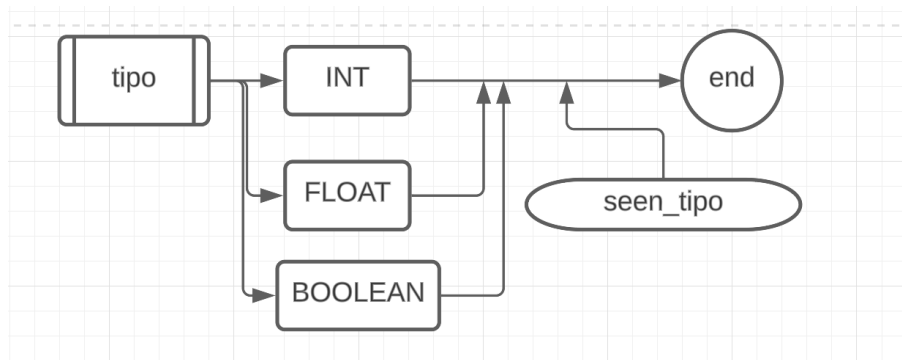
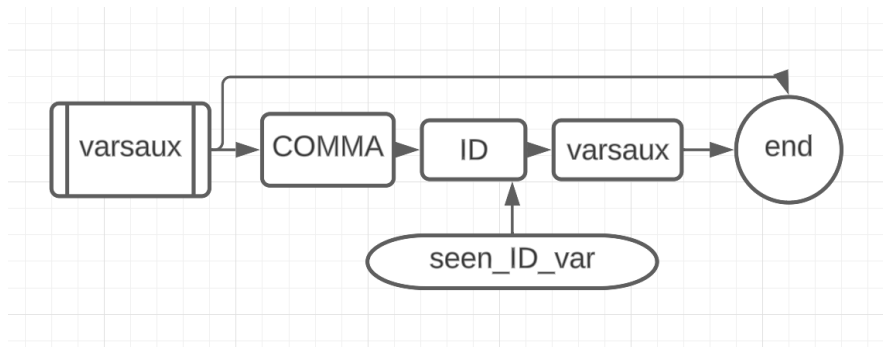
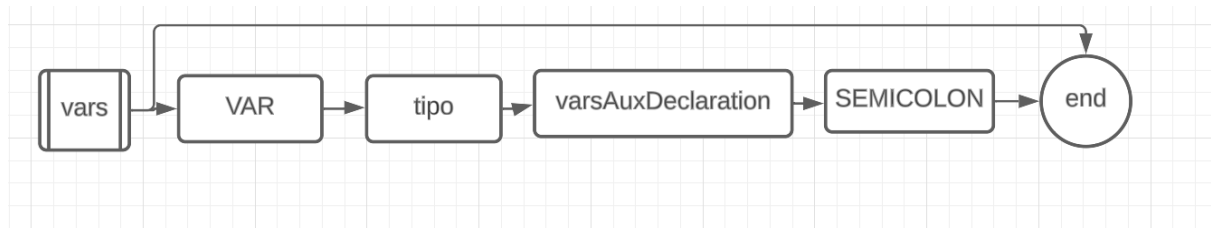
Diagramas de sintaxis con puntos neurálgicos:

A continuación se describirán los diagramas de la sintaxis creada, junto con sus puntos neurálgicos.

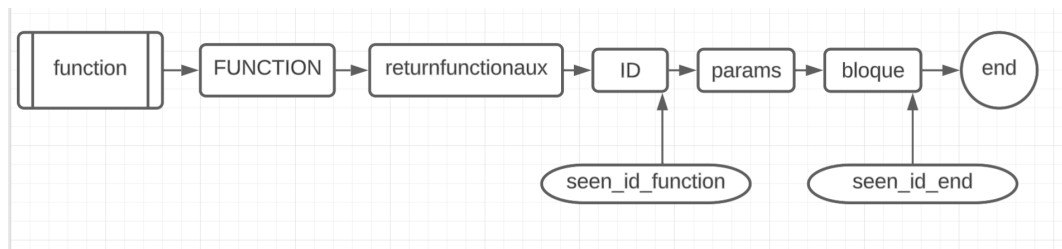
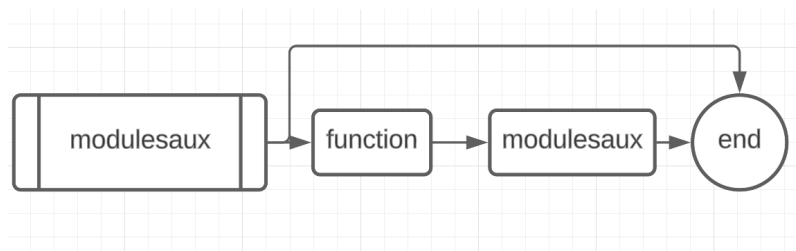
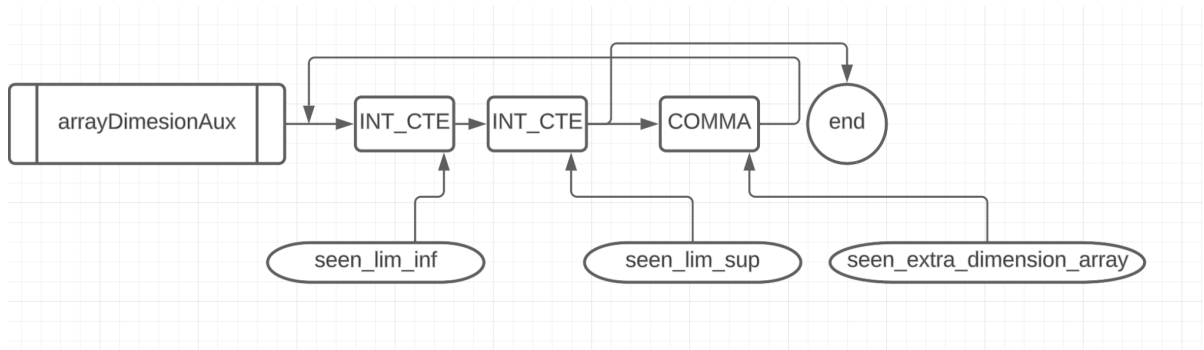
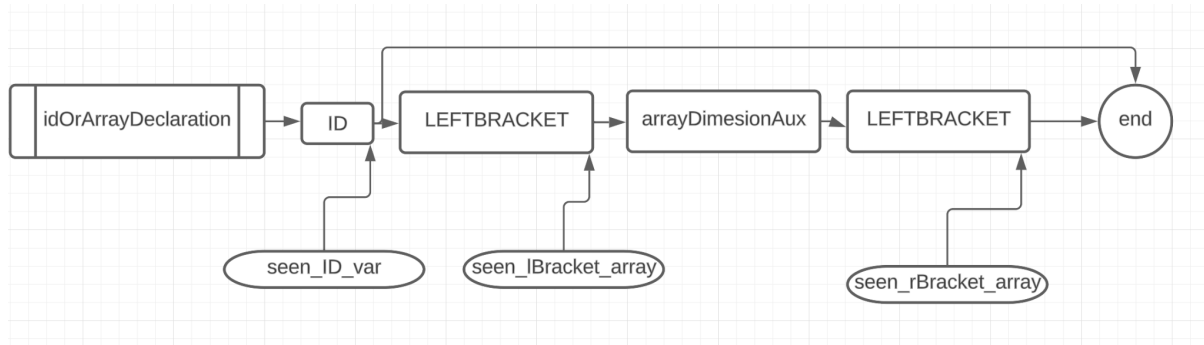
Cada punto neurálgico empieza con la palabra ‘seen\_’. Su funcionamiento se describirá brevemente al terminar los diagramas.

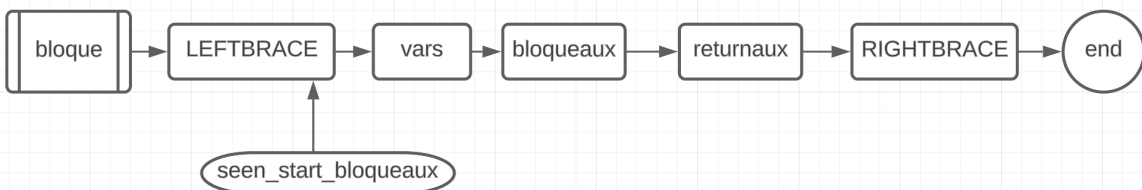
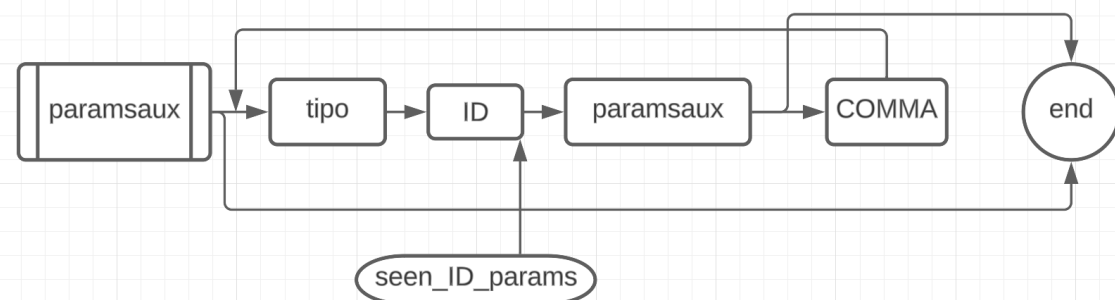
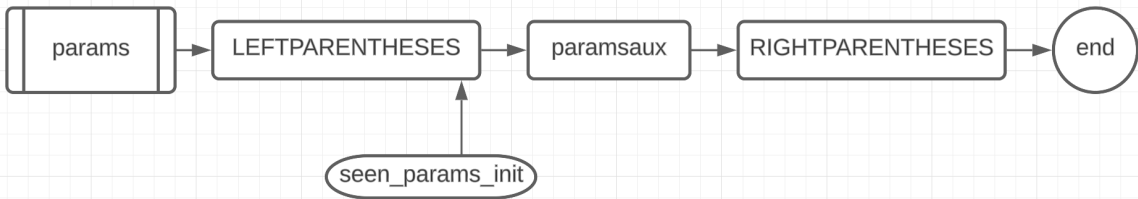
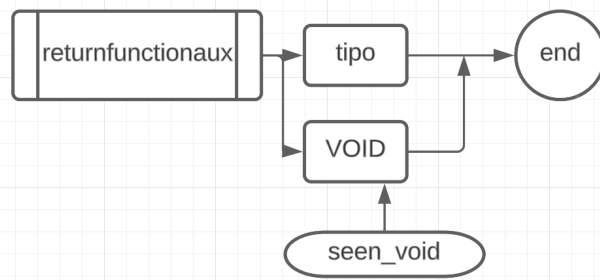
Cada diagrama contiene su nombre de sintaxis en su primer cuadrado.

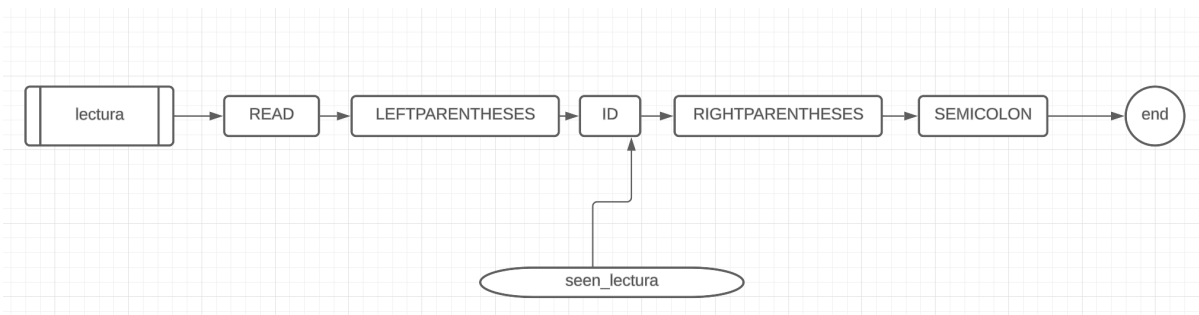
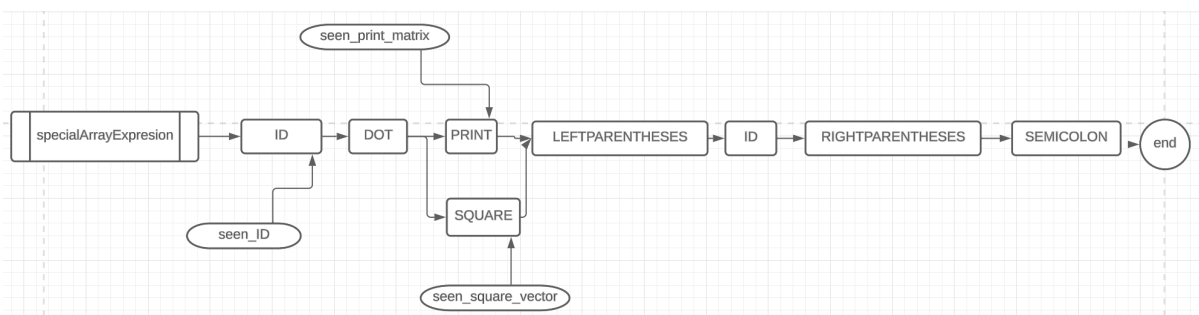
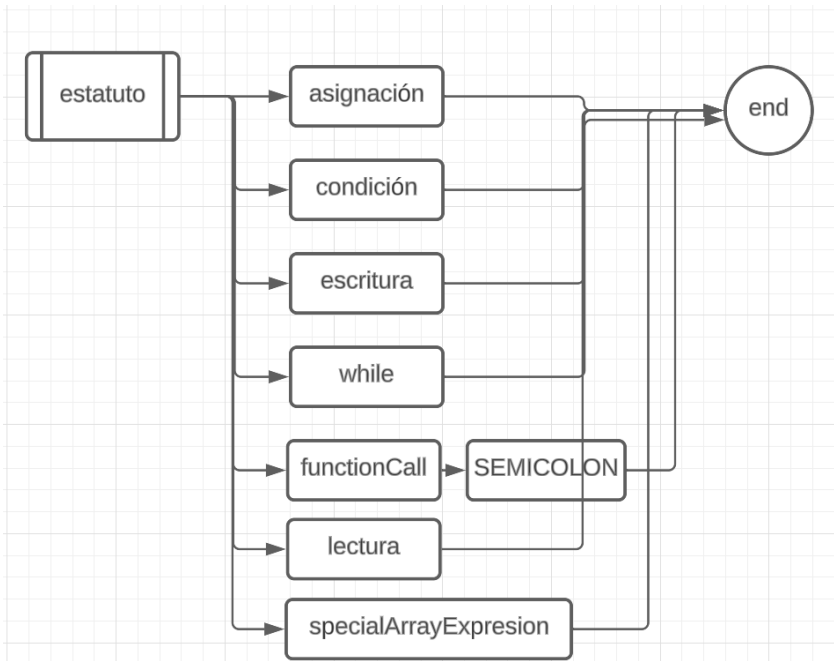
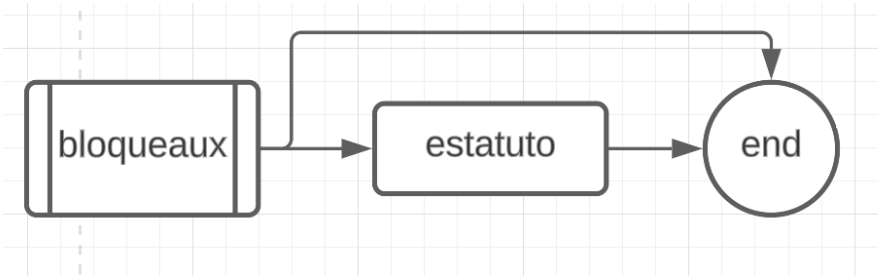
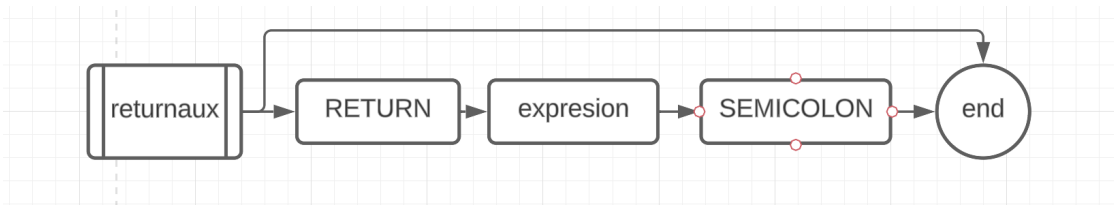


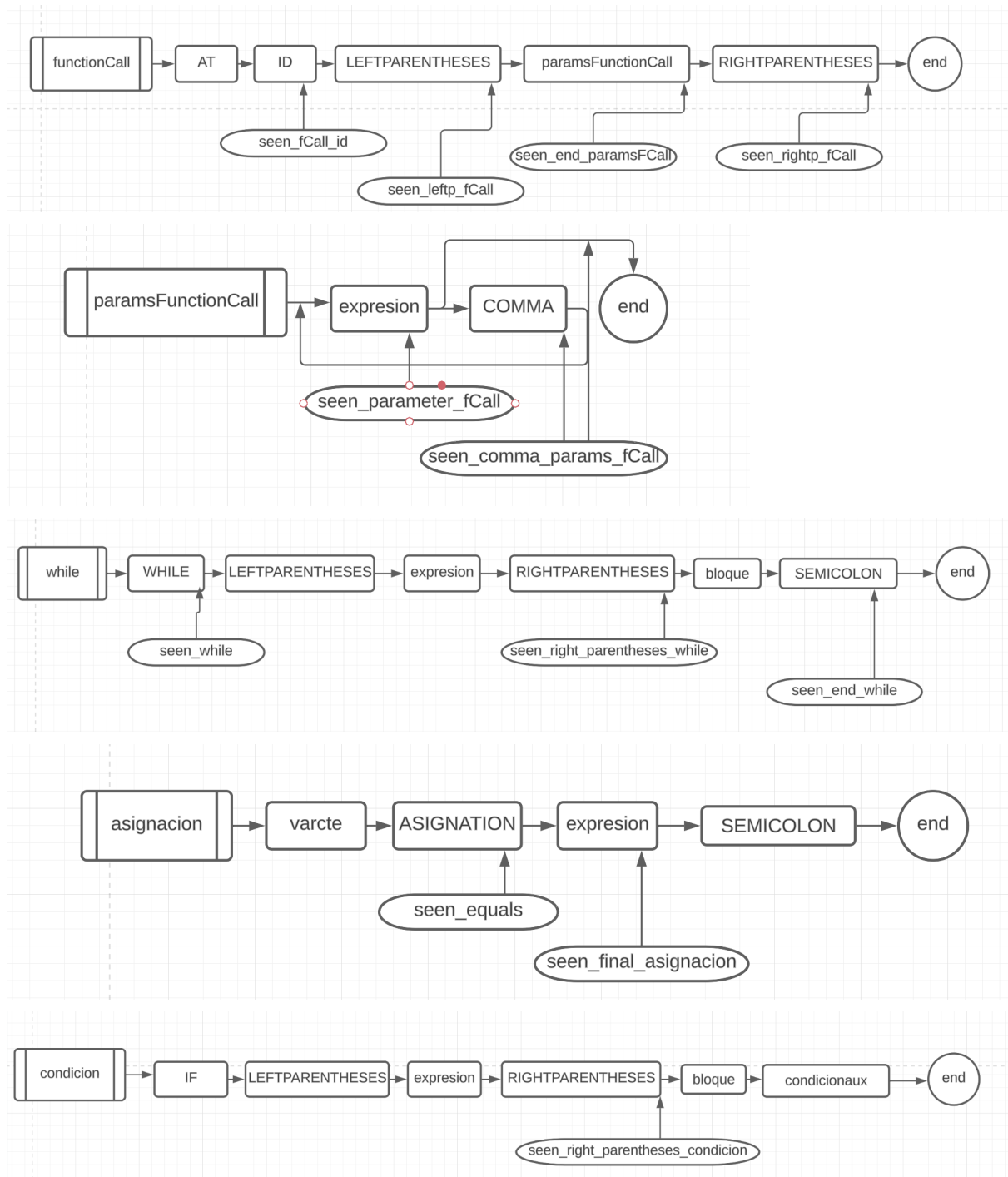


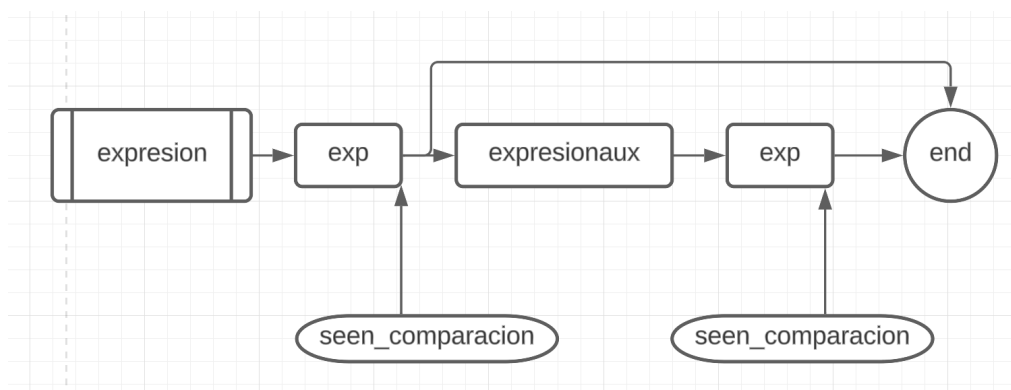
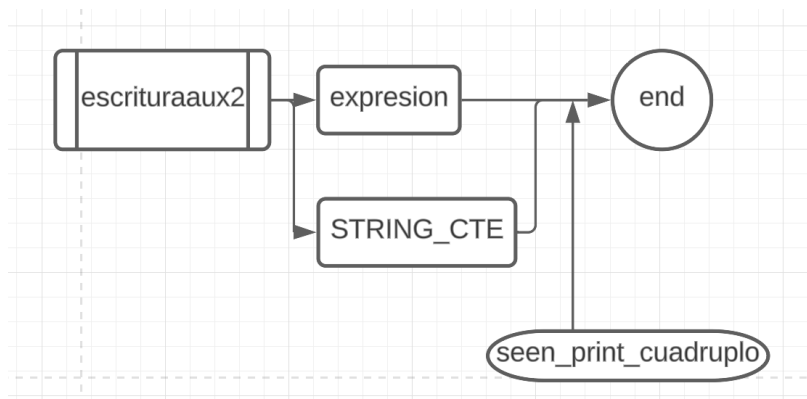
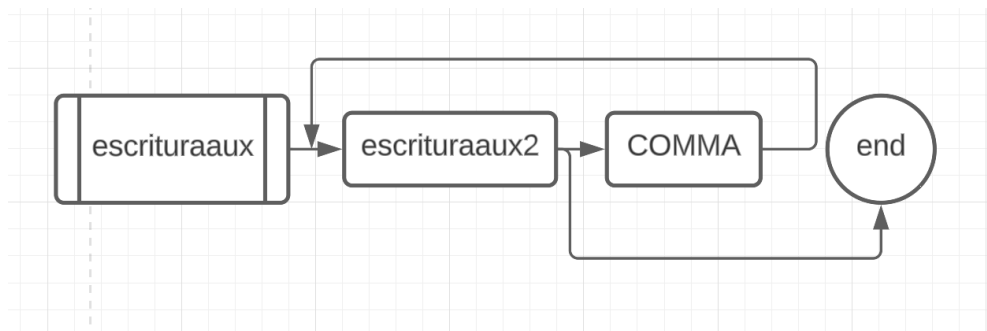
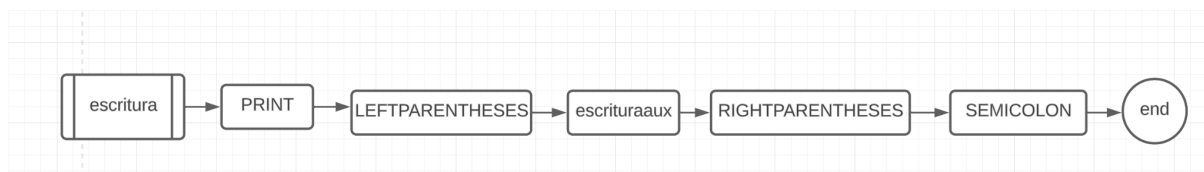
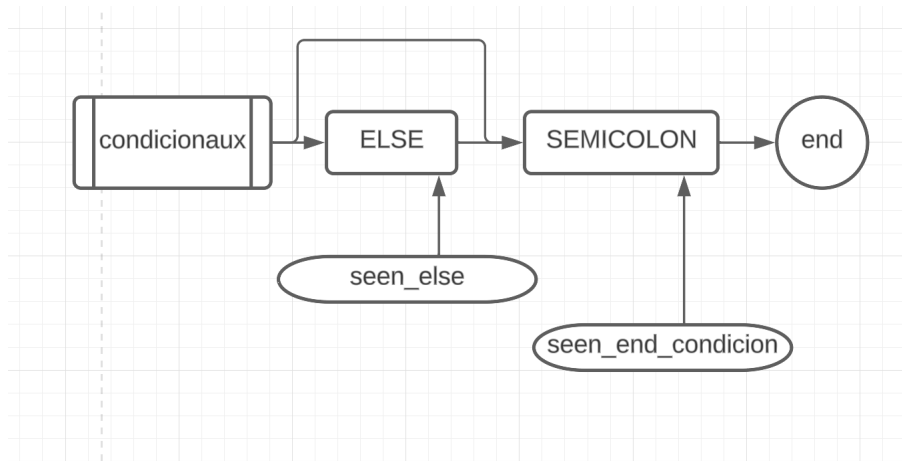


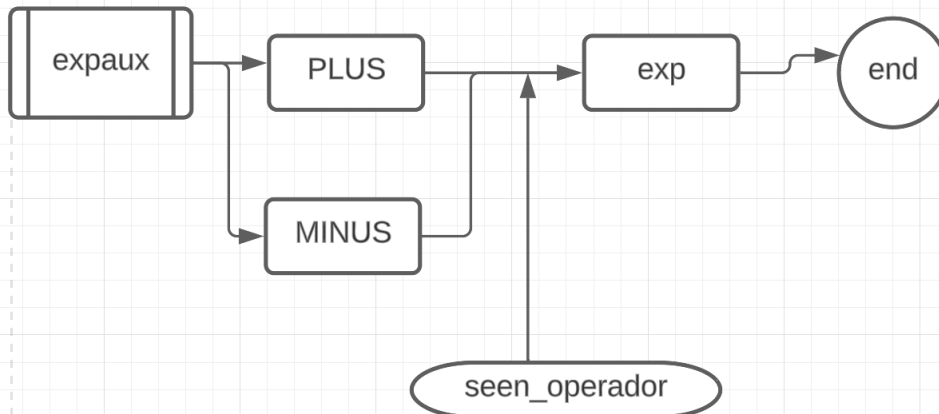
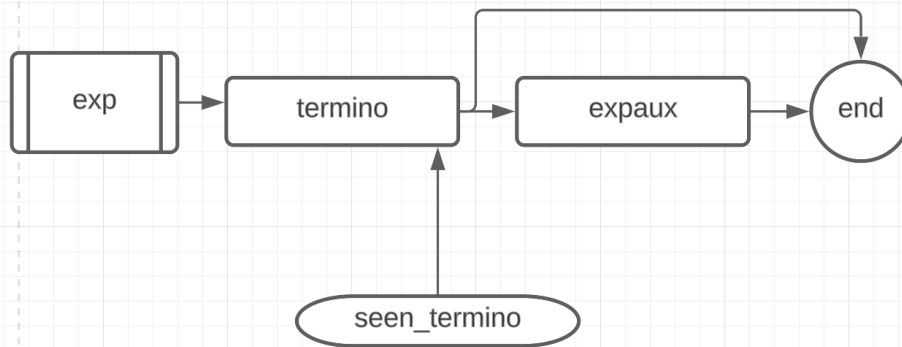
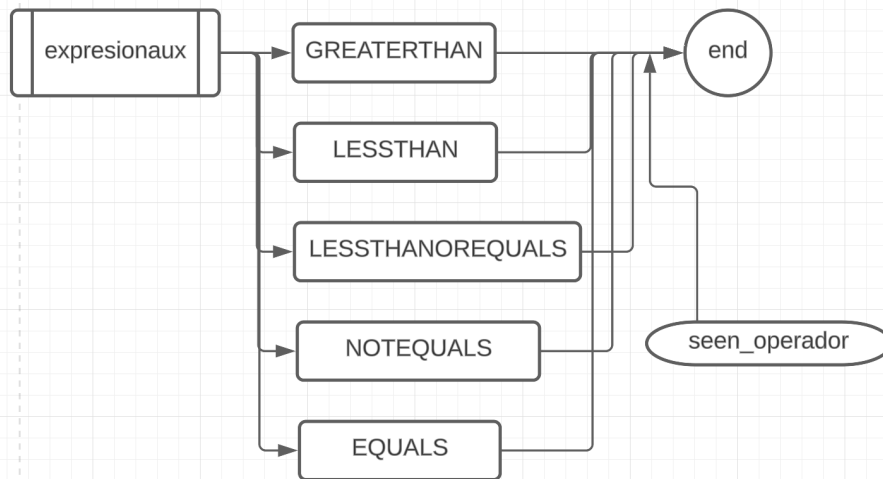


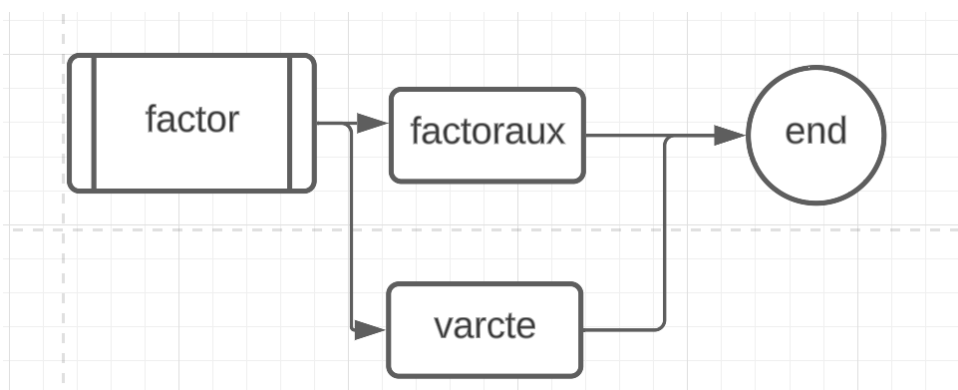
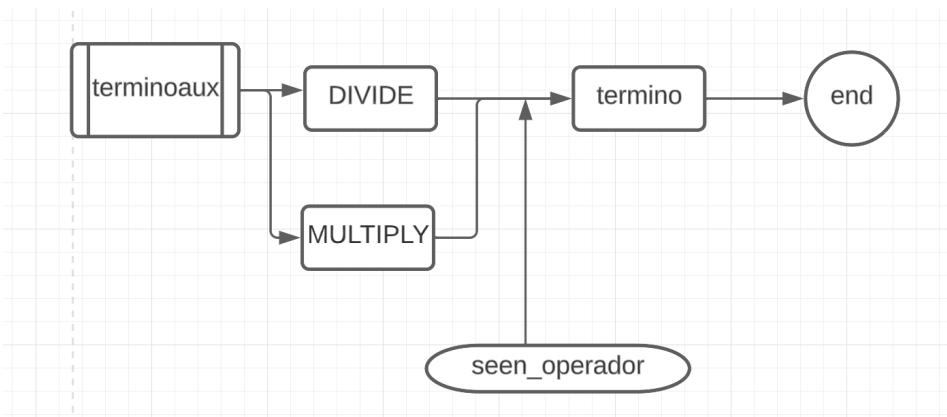
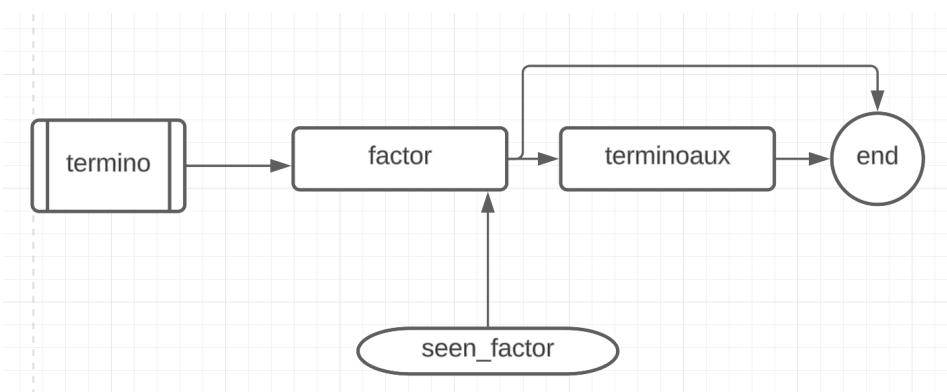


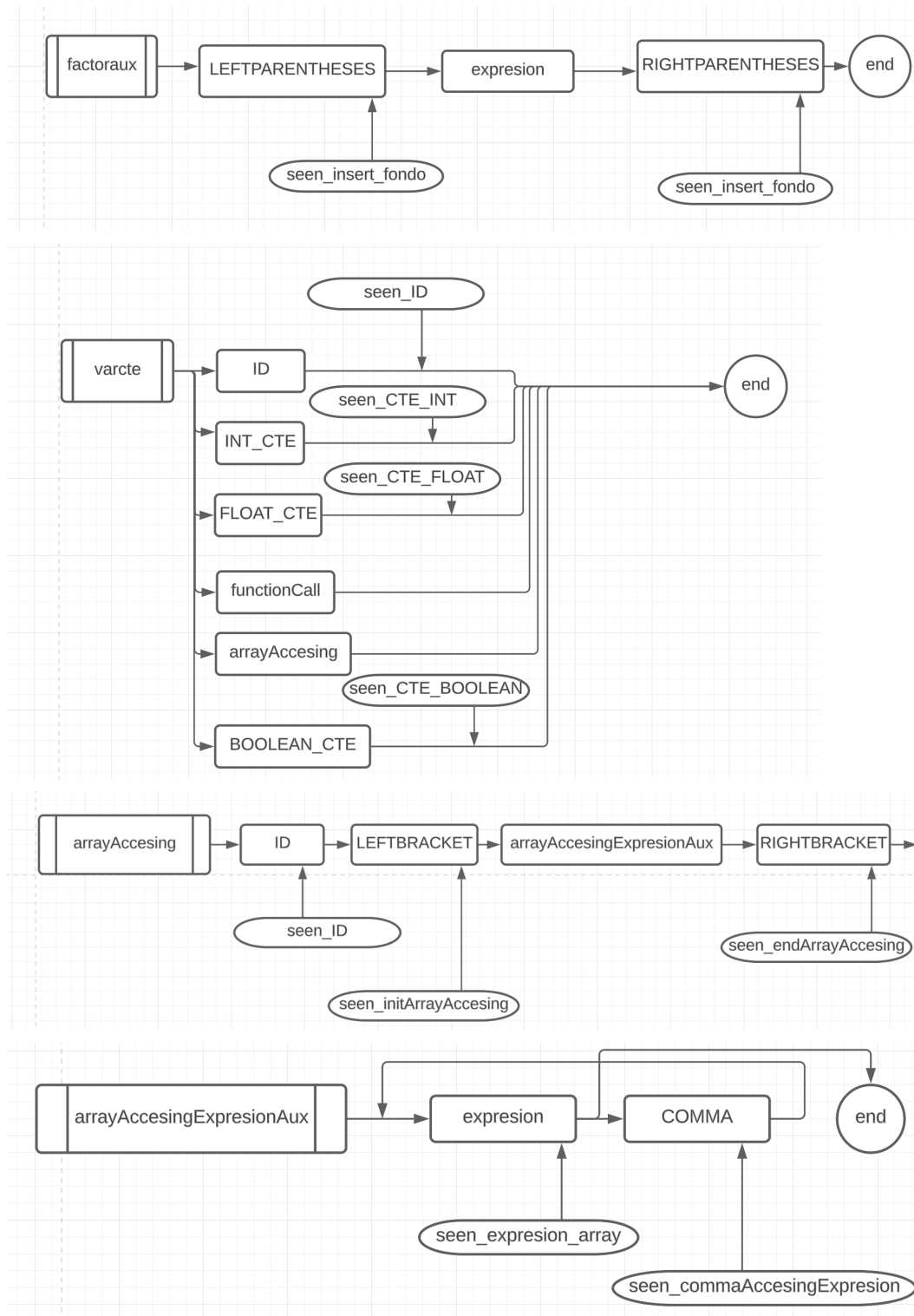












## Descripción de puntos neurálgicos:

### seen\_program:

Se guarda el nombre del programa

Se setea el nombre del programa como `currentFunction`



Se agrega a directorio de funciones el programa y se inicializa tabla de variables  
Se crea GOTO a MAIN

**seen\_function\_main:**

Se setea el nombre de la función como currentFunction  
Se agrega función a directorio de funciones de memoria de compilación  
Se rellena el primer GOTO a MAIN con el currentQuadCounter

**seen\_end\_program:**

Se agrega cuádruplo de ENDPROGRAM

**seen\_ID\_var:**

Se actualiza el currentVarId al recientemente visto  
Se agrega la variable vista a tabla de variables correspondiente, con su dirección y tipo correspondiente.  
Si es un parámetro de función, se agrega el currentType a parametersTable de la currentFunction

**seen\_tipo:**

Se actualiza el currentType al tipo recientemente visto.

**seen\_end\_declaration**

Se actualiza el currentVarId a un string vacío

**seen\_lBracket\_array:**

currentVarId se inicializa para ser array, se agrega dimension nodes a la variable  
Se setea isArray True  
Se inicializa DIM a 1  
Se inicializa R a 1

**seen\_rBracket\_array**

Se calcula cada valor de m para cada dimension node de la currentVarId

**seen\_lim\_inf**

Se setea el valor de límite inferior para la currentVarId

**seen\_lim\_sup**

Se setea el valor de límite superior para la currentVarId  
Se calcula el valor de R para la currentVarId

**seen\_extra\_dimension\_array**

Se incrementa la dimensión del currentVarId  
Se agrega un dimension node vacío a la currentVarId

**seen\_id\_function**

Se inicializa función en directorio de funciones  
Se actualiza currentFunction a la recientemente vista

**seen\_function\_end**

Crea cuádruplo que termina la función

**seen\_void**

Se setea el currentType a VOID

**seen\_params\_init**

Agrega parametersTable a currentFunction

**seen\_ID\_params**

Se actualiza el currentVarId al recientemente visto

Se agrega la variable vista a tabla de variables correspondiente, con su dirección y tipo correspondiente.

Si es un parámetro de función, se agrega el currentType a parametersTable de la currentFunction

**seen\_start\_bloqueaux**

Se agrega dirección de inicio de la función al directorio de funciones usando el valor de contador de cuádruplos

**seen\_return\_function**

Guarda último valor de expresión en la variable global de la función

Valida que la función tenga un valor de regreso, que no sea VOID

**seen\_fCall\_id**

Se guarda fondo falso en stack de operadores para proteger parámetros de la función

Se agrega functionName a stack de llamadas de funciones

**seen\_leftp\_fCall**

Crea cuádruplo ERA de la función llamada

**seen\_end\_paramsFCall**

Checa que todos los parámetros se hayan asignado antes de saltar a la función

**seen\_rightp\_fCall**

Maneja el final de una llamada a función

Crea cuádruplo GOSUB

Si la función regresa un valor, después del GOSUB guardamos en un temporal el valor de la variable global de dicha función.

Se elimina el fondo falso creado anteriormente al llamarla.

**seen\_parameter\_fCall**

Se crea cuádruplo PARAMETER con el resultado de la última expresión para asignar en el parámetro siguiente

**seen\_comma\_params\_fCall**

Se incrementa el contador de parámetros de la última llamada a función

**seen\_while**

Se agrega al stack de saltos el current quad counter

**seen\_right\_parentheses\_while**

Se checa el stack de tipos para que el último resultado sea un booleano

Se prepara cuádruplo de salto en falso necesario para el while

Se deja migaja para volver a llenar este salto en falso cuando sea necesario

**seen\_end\_while**

Se maneja final de while

Se rellena salto en falso creado al principio

Se crea GOTO para volver a calcular expresiones del while

**seen\_equals**

Se agrega operador = al stack de operadores

**seen\_final\_asignacion**

Se crea cuádruplo de asignación validando tipos

**seen\_right\_parentheses\_condicion**

Se prepara para iniciar IfElse

Se crea GOTO

Se valida que resultado de última expresión sea booleano

**seen\_else**

Se crea GOTO para saltar estatutos de ELSE

Se rellena GOTO de else para saltar directamente al espacio de Else

**seen\_end\_condicion**

Se rellena GOTO creado al momento de ver el ELSE para saltar estatutos de ELSE

**seen\_print\_cuadрупlo**

Se crea cuádruplo de PRINT con el resultado de la última expresión

**seen\_comparacion**

Se checa que no quede pendiente una comparación en la lista de operadores

Si queda pendiente una comparación, crea el cuádruplo correspondiente con los últimos dos resultados de expresiones generadas

Se comprueba compatibilidad de tipos en esta operación con cubo semántico

**seen\_operador**

Se agrega operador a pila de operadores

**seen\_termino**

Se checa que no quede pendiente una suma o resta en la lista de operadores

Si queda pendiente una suma o resta, crea el cuádruplo correspondiente con los últimos dos resultados de expresiones generadas

Se comprueba compatibilidad de tipos en esta operación con cubo semántico

**seen\_factor**

Se checa que no quede pendiente una multiplicación o división en la lista de operadores

Si queda pendiente una multiplicación o división, crea el cuádruplo correspondiente con los últimos dos resultados de expresiones generadas

Se comprueba compatibilidad de tipos en esta operación con cubo semántico

**seen\_insert\_fondo**

Se inserta fondo falso en lista de operadores para proteger operaciones con menor prioridad

**seen\_ID**

Se agrega ID a su respectiva tabla de variables con su respectiva dirección y tipo

**seen\_CTE\_INT**

Se agrega constante entera a tabla de constantes

**seen\_CTE\_FLOAT**

Se agrega constante flotante a tabla de constantes

**seen\_CTE\_BOOLEAN**

Se agrega constante booleana a tabla de constantes

**seen\_initArrayAccesing**

Se prepara accesamiento a array

Se setea Dimension a 1

Se agrega a stack de dimensiones el currentVarId y la currentDimention

Se agrega fondo falso a stack de operadores

**seen\_endArrayAccesing**

Se agregan cuádruplos y operaciones necesarias para 'flatten' array y sacar la dirección equivalente

Se crea apuntador con la dirección que contiene el valor del accesamiento y se agrega a stack de operandos

Se hace pop del stack de dimensiones

**seen\_expresion\_array**

Se agregan cuádruplos y operaciones necesarias para 'flatten' array y sacar la dirección equivalente

**seen\_commaAccesingExpresion**

Se incrementa dimensión de array de currentVarId de tope de stack de dimensiones

**Manejo de tipos, Cubo Semántico:**

A continuación se encuentra el cubo semántico, estructura utilizada para conseguir el tipo resultante de una operación tomando en cuenta los dos tipos involucrados de la operación

Tipo1	Tipo2	+	-	*	/	==	!=	<=	>=	<	>	=
int	int	int	int	int	int	boolean	boolean	boolean	boolean	boolean	boolean	sí
int	float	float	float	float	float	boolean	boolean	error	error	error	error	error
int	string	error	error	error	error	boolean	boolean	error	error	error	error	error
int	boolean	error	error	error	error	boolean	boolean	error	error	error	error	error
float	float	float	float	float	float	boolean	boolean	boolean	boolean	boolean	boolean	sí
float	string	error	error	error	error	boolean	boolean	error	error	error	error	error
float	boolean	error	error	error	error	boolean	boolean	error	error	error	error	error
string	string	error	error	error	error	boolean	boolean	error	error	error	error	sí
string	boolean	error	error	error	error	error	error	error	error	error	error	error
boolean	boolean	error	error	error	error	boolean	boolean	error	error	error	error	sí

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

A continuación se describen clases, variables y estructuras de datos utilizadas durante el momento de compilación para gestionar la memoria y parámetros de semántica.

Para agrupar datos como el directorio de funciones, se decidió utilizar un diccionario en python sobre un arreglo, esto porque nos permitía crecer de manera infinita, limitado por los recursos de la memoria donde se corre el programa de python.

Al utilizar un diccionario en vez de un arreglo, también nos permite utilizar keys para acceder más rápidamente al valor, para el caso de diccionario de funciones, cada key es el nombre de dicha función

Nombre	Descripción
--------	-------------

VirtualMemory	Clase que se encarga de asignar direcciones a variables declaradas
VirtualMemory.constantDirectory	Diccionario de python donde guardo pares de keys y values. La key siendo el valor de mi constante y value su address. Se decidió un diccionario para fácil acceso, la key es el valor de la constante y su value el address.
VirtualMemory.globalIntCounter VirtualMemory.globalFloatCounter VirtualMemory.globalBooleanCounter VirtualMemory.globalCharCounter  VirtualMemory.localIntCounter VirtualMemory.localFloatCounter VirtualMemory.localBooleanCounter VirtualMemory.localCharCounter  VirtualMemory.temporalIntCounter VirtualMemory.temporalFloatCounter VirtualMemory.temporalBooleanCounter VirtualMemory.temporalCharCounter  VirtualMemory.constantCounter	Contadores para tipos de variables con diferente scope y diferente tipo, a este contador se le suma la dirección base de su respectivo tipo y scope para conseguir la siguiente dirección disponible
VirtualMemory.setConstantInVirtualMemory(constantId)	Método que recibe como parámetro una constantId, si ya existe en el directorio de constantes regresa su dirección, si no genera su address, la guarda en directorio de constantes y regresa su dirección
VirtualMemory.dumpLocalVirtualMemory()	Método que reinicia el contador de variables locales y temporales
VirtualMemory.getNextAddressAvailable(scope, type)	Método que recibe un scope y un tipo y te regresa la siguiente dirección disponible
CompilerManager	Clase que lleva el control de directorio de funciones, stacks de operadores, operandos, chequeo de tipos
CompilerManager.functionDirectory	Diccionario responsable de almacenar toda la información necesaria de cada función, Se definió que sea un diccionario por su fácil indexamiento, se puede obtener el valor de una función en particular utilizando el nombre de la función como key
CompilerManager.functionDirectory.functionName.type	Guarda el tipo que regresa la función
CompilerManager.functionDirectory.functionName.scope	Guarda el scope de la función

CompilerManager.functionDirectory.functionName.paramsTable	Array que contiene los tipos necesarios de los parámetros de esta función, el primer índice equivale al primer parámetro etc.
CompilerManager.functionDirectory.functionName.varsTable	Diccionario que contiene las variables y su información de cada función
CompilerManager.functionDirectory.functionName.varsTable.varId	Objeto que contiene información de una variable en específico
CompilerManager.functionDirectory.functionName.varsTable.varId.address	Dirección de la variable del objeto que pertenece
CompilerManager.functionDirectory.functionName.varsTable.varId.dimensionNodes	Arreglo que contiene los nodos de dimensiones si aplica
CompilerManager.functionDirectory.functionName.varsTable.varId.dimensionNodes[DIM].lowerLimit	Límite inferior de current dimensión de un arreglo
CompilerManager.functionDirectory.functionName.varsTable.varId.dimensionNodes[DIM].upperLimit	Límite superior de current dimensión de un arreglo
CompilerManager.functionDirectory.functionName.varsTable.varId.dimensionNodes[DIM].m	M calculada para este dimensionNode en particular
CompilerManager.functionDirectory.functionName.varsTable.varId.dimensionNodes[DIM].isFinal	Booleano que indica si este dimensionNode es el último
CompilerManager.quadruples	Array que contiene los cuádruplos generados
CompilerManager.operatorsStack	Pila auxiliar para manejar operadores. Se definió una lista de python ya que se puede simular como una pila con métodos pop y append
CompilerManager.operandsStack	Pila auxiliar para manejar operandos. Se definió una lista de python ya que se puede simular como una pila con métodos pop y append
CompilerManager.typesStack	Pila auxiliar para manejar tipos. Se definió una lista de python ya que se puede simular como una pila con métodos pop y append
CompilerManager.jumpsStack	Pila auxiliar para manejar saltos. Se definió una lista de python ya que se puede simular como una pila con métodos pop y append

CompilerManager.dimensionsStack	Pila auxiliar para manejar dimensiones a la hora de acceder en arrays. Se definió una lista de python ya que se puede simular como una pila con métodos pop y append
CompilerManager.paramsVm	Clase instanciada para guardar información de programa y de cada función que será mandada como parámetro a la virtual machine, su estructura se declara en la próxima sección de memoria de ejecución
ParamsVm	Clase que guarda el tamaño de Global, Locales y Constantes, creada al final de compilación que se pasará a momento de ejecución. Se decidió crear una nueva clase para solo guardar la información relevante y no mandar información redundante
ParamsVm.GlobalSize	Diccionario que guarda tamaño y direcciones de memoria necesarios para global scope
ParamsVm.GlobalSize.size	Total de variables necesarias para global scope
ParamsVm.GlobalSize.arrayAddresses	Array que contiene todas las direcciones necesarias obtenidas en etapa de compilación
ParamsVm.MainSize	Diccionario que guarda tamaño y direcciones de memoria necesarios para main scope.
ParamsVm.MainSize.size	Total de variables necesarias para global scope
ParamsVm.MainSize.arrayAddresses	Array que contiene todas las direcciones necesarias obtenidas en etapa de compilación
ParamsVm.Constants	Diccionario que guarda tamaño y direcciones de memoria necesarios para constantes
ParamsVm.Constants.size	Número de constantes declaradas
ParamsVm.Constants.constantDictionary	Diccionario de constantes, key: constant, value: value
ParamsVm.FunctionsSize	Diccionario que guarda diccionarios de tamaño y direcciones de memoria necesarios para cada local scope. Se decidió diccionario para que pueda crecer y



	buscar más rápido por functionName
ParamsVm.FunctionsSize.functionName.size	Total de variables necesarias para local scope
ParamsVm.FunctionsSize.functionName.arrayAddresses	Array que contiene todas las direcciones necesarias obtenidas en etapa de compilación
ParamsVm.addConstant(constantId, address)	Recibe constantId y address y lo agrega a diccionario de constantes
ParamsVm.addToFunctionSize(scope, functionName, size, addresses)	Método que recibe addresses y las agrega a el scope y functionName necesario si es que aplica

Function Directory
<pre> functionName: string type: string scope: string varsTable: {   varId: {     address: number     dimationNodes: Array&lt;{       upperLimit: number,       lowerLimit: number,       m: number,       isFinal: boolean     }&gt;   } } </pre>

```

{
  "patito": {
    "type": "Program",
    "scope": "Global",
    "varsTable": {
      "A": {
        "type": "int",
        "address": 0,
        "dimentionNodes": [
          {
            "lowerLimit": 1,
            "upperLimit": 2,
            "m": 8.0,
            "isFinal": false
          },
          {
            "lowerLimit": 3,
            "upperLimit": 6,
            "m": 2.0,
            "isFinal": false
          },
          {
            "lowerLimit": 0,
            "upperLimit": 1,
            "m": -14.0,
            "isFinal": true
          }
        ]
      },
      "b": {
        "type": "int",
        "address": 16
      }
    },
    "size": { ... }
  },

```

```

Python
ParamsVm:
GlobalSize {'size': 0, 'arrayAddresses': []}
FunctionsSize {'uno': {'size': 1, 'arrayAddresses': [1001]}}
Constants {'size': 26, 'constantDictionary': {3001: 0, 3002: 10, 3003: -0.0,
3004: 1002, 3005: 1, 3006: 12, 3007: 2, 3008: 13, 3009: 3, 3010: 16, 3011: 4,
3012: 11, 3013: 5, 3014: 19, 3015: 6, 3016: 30, 3017: 7, 3018: 25, 3019: 8,
3020: 61, 3021: 9, 3022: 123, 3023: 1235, 3024: 9999, 3025: 'False', 3026:
'True'}}
MainSize {'size': 42, 'arrayAddresses': [1501, 1001, 1002, 1003, 1004, 1005,
1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 2001, 2002, 2003, 2004,
2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
2018, 2019, 2020, 2021, 2022, 2501, 2023, 2024, 2502, 2025]}

```

## DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

### Equipo de cómputo, lenguaje y utilerías especiales usadas

Esta máquina virtual se realizó en un ambiente Windows y MacOS a la vez, utilizando el IDE Pycharm tanto para escribir el compilador tanto como para ejecutarlo.

El lenguaje donde se realizó fue python, gracias a su facilidad de entendimiento tanto como su gran repertorio de librerías, se utilizó PLY (<https://www.dabeaz.com/ply/ply.html>) una librería para generar léxica y semántica con código de python.

Como manejador de versiones se utilizó Git.

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

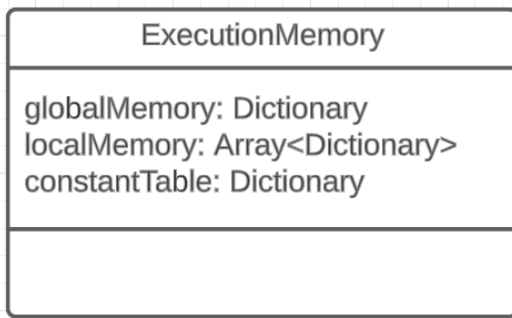
Al iniciar nuestro proceso de ejecución, lo primero que hacemos es 'vaciar' el contenido de ParamsVm en nuestra executionMemory, asumiendo que Main no se puede volver a llamar por lo que es fijo.

Nuestro cambio de contexto por parte de funciones lo manejamos con un stack, iniciamos con Main al inicio del stack y vamos agregando contextos cuando se van llamando funciones, haciendo pop cuando se termina la llamada de esta función

Nombre	Descripción
ExecutionMemory.globalMemory	ObjDicDiccionario que contiene las direcciones como key y su valor como keyValue. Se instancia al crear la clase ExecutionMemory ya que desde primer momento sabes el tamaño total de este scope. Se decidió un diccionario ya que acceder rápidamente al valor de las direcciones utilizando la address como key.
ExecutionMemory.constantTable	Diccionario que contiene las constantes junto con su dirección para poder traducirlo al momento de su ejecución
ExecutionMemory.localMemory	Array que contiene los contextos de memoria de cada función llamada, véase contexto como un diccionario de direcciones con su valor. Se decidió un array para simular un stack.
ExecutionMemory.addLocalMemory(functionName)	Toma el tamaño total de la función con el nombre de la función recibido y lo agrega a la pila de contextos, creando un diccionario con las variables necesarias para su mapeo de memoria.
ExecutionMemory.setValueToAddress(value, address)	Método que busca el address en contexto local, si no en global y setea su valor
ExecutionMemory.getValueOfAddress(address)	Método que busca el valor con key address en contexto local, si no en contexto global y por último en constantes, regresa el valor encontrado

Execution.localMemory es mi manejador de contextos, cada que una función es llamada se agrega al stack un objeto con las direcciones de memoria necesarias para este contexto en particular, al finalizar la llamada se libera esta memoria y se regresa al contexto anterior

Ejemplo de local memory, tomar nota que la key de cada diccionario es una dirección, y su valor el valor que guardan



Python

```
GlobalMemory {0: 2, 1: 2}
LocalMemory [{1001: 3, 2001: 2, 2002: 2}]
ConstantDictionary {3001: 1, 3002: 2, 3003: 0, 3004: 'INGRESA FIBONACCI A
CALCULAR', 3005: 'RESULTADO RECURSIVO: ', 3006: 'RESULTADO CICLICO: '}
```

## PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :

### Fibonacci recursivo y cíclico

#### Código ejecutado

Python

Programa Fibonnaci;

```
Function int fibRecursivo(int n){
    var int value;
    if(n <= 1){
        value = n;
    }else{
        value = @fibRecursivo(n - 1) + @fibRecursivo(n - 2);
    };
    return value;
}
```

```

Function int fibCiclico(int n){
    var int init, fib, aux;
    init = 1;
    fib = 0;
    aux = 1;
    while(init <= n) {
        aux = aux + fib;
        fib = aux - fib;
        init = init + 1;
    };
    return fib;
}

Main(){
    var int cont;
    print("INGRESA FIBONACCI A CALCULAR");
    read(cont);
    print("RESULTADO RECURSIVO: ", @fibRecursivo(cont));
    print("RESULTADO CICLICO: ", @fibCiclico(cont));
}

```

Cuádruplos generados:

```

Python
0 []
1 [GOTO, '', '', 34]
2 ['<=', 1001, 3001, 2501]
3 [GOTO, 2501, '', 6]
4 ['=', 1001, '', 1002]
5 [GOTO, '', '', 18]
6 [ERA, '', '', 'fibRecursivo']
7 ['- ', 1001, 3001, 2001]
8 [PARAMETER, '', 2001, 0]
9 [GOSUB, 0, '', 2]
10 ['=', 0, '', 2002]
11 [ERA, '', '', 'fibRecursivo']
12 ['- ', 1001, 3002, 2003]
13 [PARAMETER, '', 2003, 0]
14 [GOSUB, 0, '', 2]
15 ['=', 0, '', 2004]
16 ['+', 2002, 2004, 2005]
17 ['=', 2005, '', 1002]

```

```
18 ['=', 1002, '', 0]
19 [ENDFUNC, '', '', '']
20 ['=', 3001, '', 1002]
21 ['=', 3003, '', 1003]
22 ['=', 3001, '', 1004]
23 ['<=', 1002, 1001, 2501]
24 [GOTO, 2501, '', 32]
25 ['+', 1004, 1003, 2001]
26 ['=', 2001, '', 1004]
27 ['-', 1004, 1003, 2002]
28 ['=', 2002, '', 1003]
29 ['+', 1002, 3001, 2003]
30 ['=', 2003, '', 1002]
31 [GOTO, '', '', 23]
32 ['=', 1003, '', 1]
33 [ENDFUNC, '', '', '']
34 [PRINT, '', '', 3004]
35 [READ, '', '', 1001]
36 [PRINT, '', '', 3005]
37 [ERA, '', '', 'fibRecursivo']
38 [PARAMETER, '', 1001, 0]
39 [GOSUB, 0, '', 2]
40 ['=', 0, '', 2001]
41 [PRINT, '', '', 2001]
42 [PRINT, '', '', 3006]
43 [ERA, '', '', 'fibCiclico']
44 [PARAMETER, '', 1001, 0]
45 [GOSUB, 1, '', 20]
46 ['=', 1, '', 2002]
47 [PRINT, '', '', 2002]
48 [END, '', '', '']
```

## Resultado

```
INGRESA FIBONACCI A CALCULAR
8
RESULTADO RECURSIVO:
21
RESULTADO CICLICO:
21
```

## Factorial Cíclico y Recursivo

### Código ejecutado

```
Python
Programa Factorial;

Function int factorialRecursivo(int n){
    var int value;
    if(n == 1){
        value = 1;
    }else{
        value = n * @factorialRecursivo(n-1);
    };
    return value;
}

Function int factorialCiclico(int n){
    var int aux, i;
    i = 1;
    aux = 1;
    while(i <= n){
        aux = aux * i;
        i = i + 1;
    };
    return aux;
}
```

```

}

Main(){
    var int cont;
    print("INGRESA FACTORIAL A CALCULAR");
    read(cont);
    print("RESULTADO RECURSIVO: ", @factorialRecursivo(cont));
    print("RESULTADO RECURSIVO: ", @factorialCiclico(cont));
}

```

Cuádruplos generados:

```

Python
0 []
1 [GOTO, '', '', 26]
2 ['==', 1001, 3001, 2501]
3 [GOTOIF, 2501, '', 6]
4 ['=', 3001, '', 1002]
5 [GOTO, '', '', 13]
6 [ERA, '', '', 'factorialRecursivo']
7 ['- ', 1001, 3001, 2001]
8 [PARAMETER, '', 2001, 0]
9 [GOSUB, 0, '', 2]
10 ['=', 0, '', 2002]
11 ['*', 1001, 2002, 2003]
12 ['=', 2003, '', 1002]
13 ['=', 1002, '', 0]
14 [ENDFUNC, '', '', '']
15 ['=', 3001, '', 1003]
16 ['=', 3001, '', 1002]
17 ['<=', 1003, 1001, 2501]
18 [GOTOIF, 2501, '', 24]
19 ['*', 1002, 1003, 2001]
20 ['=', 2001, '', 1002]
21 ['+', 1003, 3001, 2002]
22 ['=', 2002, '', 1003]
23 [GOTO, '', '', 17]
24 ['=', 1002, '', 1]
25 [ENDFUNC, '', '', '']
26 [PRINT, '', '', 3002]
27 [READ, '', '', 1001]

```

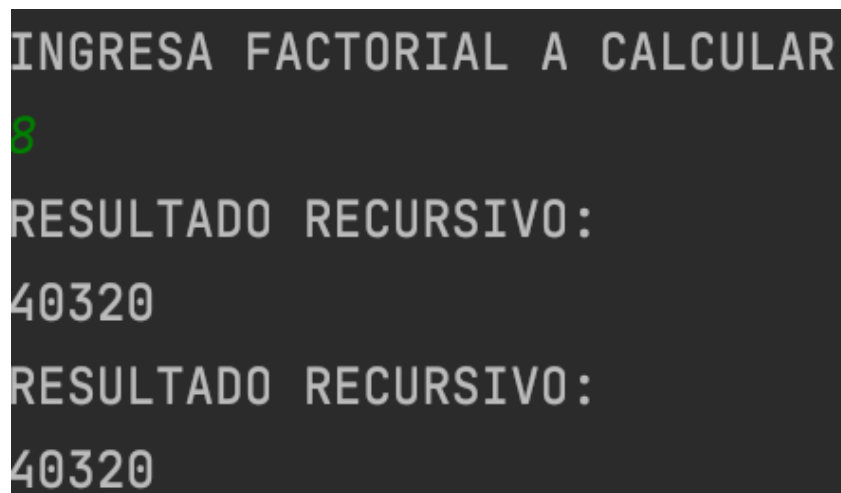


```

28 [PRINT, '', '', 3003]
29 [ERA, '', '', 'factorialRecursivo']
30 [PARAMETER, '', 1001, 0]
31 [GOSUB, 0, '', 2]
32 ['=', 0, '', 2001]
33 [PRINT, '', '', 2001]
34 [PRINT, '', '', 3003]
35 [ERA, '', '', 'factorialCiclico']
36 [PARAMETER, '', 1001, 0]
37 [GOSUB, 1, '', 15]
38 ['=', 1, '', 2002]
39 [PRINT, '', '', 2002]
40 [END, '', '', '']

```

Resultado



```

INGRESA FACTORIAL A CALCULAR
3
RESULTADO RECURSIVO:
40320
RESULTADO RECURSIVO:
40320

```

Find:

Código ejecutado

```

Python
Programa FindTest;

Main(){
    var boolean found;
    var int cont, objective, index, A[0 10], size;

    A[0] = 10;
    A[1] = 12;
    A[2] = 13;
    A[3] = 16;
    A[4] = 11;
    A[5] = 19;
    A[6] = 30;
    A[7] = 25;
    A[8] = 61;
    A[9] = 123;
    A[10] = 1235;

    print("VALOR A ENCONTRAR");
    read(objective);

    size = 10;
    index = 9999;
    found = False;
    cont = 0;

    while(cont <= size){
        if(objective == A[cont]){
            found = True;
            index = cont;
        };
        cont = cont + 1;
    };

    print(index);
}

```

Cuádruplos generados:

```

Python
0 []
1 [GOTO, '', '', 2]

```

```

2 [VERIFY, 3001, 3001, 3002]
3 ['+', 3001, 3003, 2001]
4 ['+', 2001, 3004, 2002]
5 ['=', 3002, '', '*2002']
6 [VERIFY, 3005, 3001, 3002]
7 ['+', 3005, 3003, 2003]
8 ['+', 2003, 3004, 2004]
9 ['=', 3006, '', '*2004']
10 [VERIFY, 3007, 3001, 3002]
11 ['+', 3007, 3003, 2005]
12 ['+', 2005, 3004, 2006]
13 ['=', 3008, '', '*2006']
14 [VERIFY, 3009, 3001, 3002]
15 ['+', 3009, 3003, 2007]
16 ['+', 2007, 3004, 2008]
17 ['=', 3010, '', '*2008']
18 [VERIFY, 3011, 3001, 3002]
19 ['+', 3011, 3003, 2009]
20 ['+', 2009, 3004, 2010]
21 ['=', 3012, '', '*2010']
22 [VERIFY, 3013, 3001, 3002]
23 ['+', 3013, 3003, 2011]
24 ['+', 2011, 3004, 2012]
25 ['=', 3014, '', '*2012']
26 [VERIFY, 3015, 3001, 3002]
27 ['+', 3015, 3003, 2013]
28 ['+', 2013, 3004, 2014]
29 ['=', 3016, '', '*2014']
30 [VERIFY, 3017, 3001, 3002]
31 ['+', 3017, 3003, 2015]
32 ['+', 2015, 3004, 2016]
33 ['=', 3018, '', '*2016']
34 [VERIFY, 3019, 3001, 3002]
35 ['+', 3019, 3003, 2017]
36 ['+', 2017, 3004, 2018]
37 ['=', 3020, '', '*2018']
38 [VERIFY, 3021, 3001, 3002]
39 ['+', 3021, 3003, 2019]
40 ['+', 2019, 3004, 2020]
41 ['=', 3022, '', '*2020']
42 [VERIFY, 3002, 3001, 3002]
43 ['+', 3002, 3003, 2021]
44 ['+', 2021, 3004, 2022]
45 ['=', 3023, '', '*2022']
46 [PRINT, '', '', 3024]
47 [READ, '', '', 1002]
48 ['=', 3002, '', 1015]
49 ['=', 3025, '', 1003]

```

```

50 ['=', 3026, '', 1501]
51 ['=', 3001, '', 1001]
52 ['<=', 1001, 1015, 2501]
53 [GOTO, 2501, '', 64]
54 [VERIFY, 1001, 3001, 3002]
55 ['+', 1001, 3003, 2023]
56 ['+', 2023, 3004, 2024]
57 ['==', 1002, '*2024', 2502]
58 [GOTO, 2502, '', 61]
59 ['=', 3027, '', 1501]
60 ['=', 1001, '', 1003]
61 ['+', 1001, 3005, 2025]
62 ['=', 2025, '', 1001]
63 [GOTO, '', '', 52]
64 [PRINT, '', '', 3028]
65 [PRINT, '', '', 1003]
66 [END, '', '', '']

```

Resultado:

VALOR A ENCONTRAR

16

ENCONTRADO EN INDEX:

3

## Matrix Multiplication

Código ejecutado:

```

Python
Programa Matrix;

```

```

Main(){
    var int matrix1[0 2, 0 2], matrix2[0 2, 0 2], resultMatrix[0 2, 0 2],
    size, i, j, k, sum;
    size = 3;

    matrix1[0, 0] = 1;
    matrix1[0, 1] = 2;
    matrix1[0, 2] = 3;
    matrix1[1, 0] = 4;
    matrix1[1, 1] = 5;
    matrix1[1, 2] = 6;
    matrix1[2, 0] = 7;
    matrix1[2, 1] = 8;
    matrix1[2, 2] = 9;

    matrix2[0, 0] = 9;
    matrix2[0, 1] = 8;
    matrix2[0, 2] = 7;
    matrix2[1, 0] = 6;
    matrix2[1, 1] = 5;
    matrix2[1, 2] = 4;
    matrix2[2, 0] = 3;
    matrix2[2, 1] = 2;
    matrix2[2, 2] = 1;

    i = 0;
    while(i < size){
        j = 0;
        while(j < size){
            k = 0;
            sum = 0;
            while(k < size){
                sum = sum + matrix1[i, k] * matrix2[k, j];
                k = k + 1;
            };
            resultMatrix[i, j] = sum;
            j = j + 1;
        };
        i = i + 1;
    };
    print("MATRIX1:");
    matrix1.print();
    print("MATRIX2:");
    matrix2.print();

    print("MATRIX MULTIPLICATION RESULT");
    resultMatrix.print();
}

```

```
}
```

Cuádruplos generados:

Python

```
0 []
1 [GOTO, '', '', 2]
2 ['=', 3001, '', 1028]
3 [VERIFY, 3002, 3002, 3003]
4 ['*', 3002, 3004, 2001]
5 [VERIFY, 3002, 3002, 3003]
6 ['+', 2001, 3002, 2002]
7 ['+', 2002, 3005, 2003]
8 ['+', 2003, 3006, 2004]
9 ['=', 3007, '', '*2004']
10 [VERIFY, 3002, 3002, 3003]
11 ['*', 3002, 3004, 2005]
12 [VERIFY, 3007, 3002, 3003]
13 ['+', 2005, 3007, 2006]
14 ['+', 2006, 3005, 2007]
15 ['+', 2007, 3006, 2008]
16 ['=', 3003, '', '*2008']
17 [VERIFY, 3002, 3002, 3003]
18 ['*', 3002, 3004, 2009]
19 [VERIFY, 3003, 3002, 3003]
20 ['+', 2009, 3003, 2010]
21 ['+', 2010, 3005, 2011]
22 ['+', 2011, 3006, 2012]
23 ['=', 3001, '', '*2012']
24 [VERIFY, 3007, 3002, 3003]
25 ['*', 3007, 3004, 2013]
26 [VERIFY, 3002, 3002, 3003]
27 ['+', 2013, 3002, 2014]
28 ['+', 2014, 3005, 2015]
29 ['+', 2015, 3006, 2016]
30 ['=', 3008, '', '*2016']
31 [VERIFY, 3007, 3002, 3003]
32 ['*', 3007, 3004, 2017]
33 [VERIFY, 3007, 3002, 3003]
34 ['+', 2017, 3007, 2018]
```

```

35 ['+', 2018, 3005, 2019]
36 ['+', 2019, 3006, 2020]
37 ['=', 3009, '', '*2020']
38 [VERIFY, 3007, 3002, 3003]
39 ['*', 3007, 3004, 2021]
40 [VERIFY, 3003, 3002, 3003]
41 ['+', 2021, 3003, 2022]
42 ['+', 2022, 3005, 2023]
43 ['+', 2023, 3006, 2024]
44 ['=', 3010, '', '*2024']
45 [VERIFY, 3003, 3002, 3003]
46 ['*', 3003, 3004, 2025]
47 [VERIFY, 3002, 3002, 3003]
48 ['+', 2025, 3002, 2026]
49 ['+', 2026, 3005, 2027]
50 ['+', 2027, 3006, 2028]
51 ['=', 3011, '', '*2028']
52 [VERIFY, 3003, 3002, 3003]
53 ['*', 3003, 3004, 2029]
54 [VERIFY, 3007, 3002, 3003]
55 ['+', 2029, 3007, 2030]
56 ['+', 2030, 3005, 2031]
57 ['+', 2031, 3006, 2032]
58 ['=', 3012, '', '*2032']
59 [VERIFY, 3003, 3002, 3003]
60 ['*', 3003, 3004, 2033]
61 [VERIFY, 3003, 3002, 3003]
62 ['+', 2033, 3003, 2034]
63 ['+', 2034, 3005, 2035]
64 ['+', 2035, 3006, 2036]
65 ['=', 3013, '', '*2036']
66 [VERIFY, 3002, 3002, 3003]
67 ['*', 3002, 3004, 2037]
68 [VERIFY, 3002, 3002, 3003]
69 ['+', 2037, 3002, 2038]
70 ['+', 2038, 3005, 2039]
71 ['+', 2039, 3014, 2040]
72 ['=', 3013, '', '*2040']
73 [VERIFY, 3002, 3002, 3003]
74 ['*', 3002, 3004, 2041]
75 [VERIFY, 3007, 3002, 3003]
76 ['+', 2041, 3007, 2042]
77 ['+', 2042, 3005, 2043]
78 ['+', 2043, 3014, 2044]
79 ['=', 3012, '', '*2044']
80 [VERIFY, 3002, 3002, 3003]
81 ['*', 3002, 3004, 2045]
82 [VERIFY, 3003, 3002, 3003]

```

```

83 ['+', 2045, 3003, 2046]
84 ['+', 2046, 3005, 2047]
85 ['+', 2047, 3014, 2048]
86 ['=', 3011, '', '*2048']
87 [VERIFY, 3007, 3002, 3003]
88 ['*', 3007, 3004, 2049]
89 [VERIFY, 3002, 3002, 3003]
90 ['+', 2049, 3002, 2050]
91 ['+', 2050, 3005, 2051]
92 ['+', 2051, 3014, 2052]
93 ['=', 3010, '', '*2052']
94 [VERIFY, 3007, 3002, 3003]
95 ['*', 3007, 3004, 2053]
96 [VERIFY, 3007, 3002, 3003]
97 ['+', 2053, 3007, 2054]
98 ['+', 2054, 3005, 2055]
99 ['+', 2055, 3014, 2056]
100 ['=', 3009, '', '*2056']
101 [VERIFY, 3007, 3002, 3003]
102 ['*', 3007, 3004, 2057]
103 [VERIFY, 3003, 3002, 3003]
104 ['+', 2057, 3003, 2058]
105 ['+', 2058, 3005, 2059]
106 ['+', 2059, 3014, 2060]
107 ['=', 3008, '', '*2060']
108 [VERIFY, 3003, 3002, 3003]
109 ['*', 3003, 3004, 2061]
110 [VERIFY, 3002, 3002, 3003]
111 ['+', 2061, 3002, 2062]
112 ['+', 2062, 3005, 2063]
113 ['+', 2063, 3014, 2064]
114 ['=', 3001, '', '*2064']
115 [VERIFY, 3003, 3002, 3003]
116 ['*', 3003, 3004, 2065]
117 [VERIFY, 3007, 3002, 3003]
118 ['+', 2065, 3007, 2066]
119 ['+', 2066, 3005, 2067]
120 ['+', 2067, 3014, 2068]
121 ['=', 3003, '', '*2068']
122 [VERIFY, 3003, 3002, 3003]
123 ['*', 3003, 3004, 2069]
124 [VERIFY, 3003, 3002, 3003]
125 ['+', 2069, 3003, 2070]
126 ['+', 2070, 3005, 2071]
127 ['+', 2071, 3014, 2072]
128 ['=', 3007, '', '*2072']
129 ['=', 3002, '', 1029]
130 ['<', 1029, 1028, 2501]

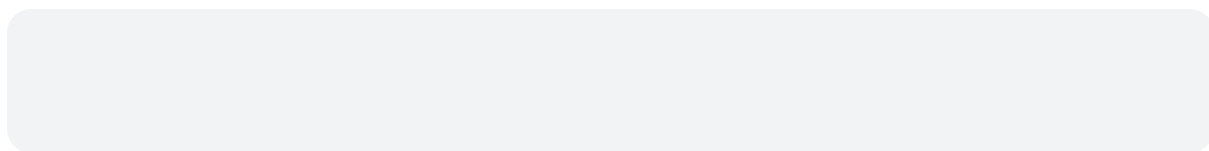
```



```

131 [GOTO, 2501, '', 170]
132 ['=', 3002, '', 1030]
133 ['<', 1030, 1028, 2502]
134 [GOTO, 2502, '', 167]
135 ['=', 3002, '', 1031]
136 ['=', 3002, '', 1032]
137 ['<', 1031, 1028, 2503]
138 [GOTO, 2503, '', 157]
139 [VERIFY, 1029, 3002, 3003]
140 ['*', 1029, 3004, 2073]
141 [VERIFY, 1031, 3002, 3003]
142 ['+', 2073, 1031, 2074]
143 ['+', 2074, 3005, 2075]
144 ['+', 2075, 3006, 2076]
145 [VERIFY, 1031, 3002, 3003]
146 ['*', 1031, 3004, 2077]
147 [VERIFY, 1030, 3002, 3003]
148 ['+', 2077, 1030, 2078]
149 ['+', 2078, 3005, 2079]
150 ['+', 2079, 3014, 2080]
151 ['*', '*2076', '*2080', 2081]
152 ['+', 1032, 2081, 2082]
153 ['=', 2082, '', 1032]
154 ['+', 1031, 3007, 2083]
155 ['=', 2083, '', 1031]
156 [GOTO, '', '', 137]
157 [VERIFY, 1029, 3002, 3003]
158 ['*', 1029, 3004, 2084]
159 [VERIFY, 1030, 3002, 3003]
160 ['+', 2084, 1030, 2085]
161 ['+', 2085, 3005, 2086]
162 ['+', 2086, 3015, 2087]
163 ['=', 1032, '', '*2087']
164 ['+', 1030, 3007, 2088]
165 ['=', 2088, '', 1030]
166 [GOTO, '', '', 133]
167 ['+', 1029, 3007, 2089]
168 ['=', 2089, '', 1029]
169 [GOTO, '', '', 130]
170 [PRINT, '', '', 3016]
171 [PRINTMATRIX, 9, 1001, '']
172 [PRINT, '', '', 3017]
173 [PRINTMATRIX, 9, 1010, '']
174 [PRINT, '', '', 3018]
175 [PRINTMATRIX, 9, 1019, '']
176 [END, '', '', '']

```



## Resultado

```
MATRIX1:
|| PRINTING MATRIX ||
1
2
3
4
5
6
7
8
9
|| END PRINTING MATRIX ||
MATRIX2:
|| PRINTING MATRIX ||
9
8
7
6
5
4
3
2
1
|| END PRINTING MATRIX ||
MATRIX MULTIPLICATION RESULT
|| PRINTING MATRIX ||
30
24
18
84
69
54
138
114
90
..
```

## Vector Manipulation

```

Python
Programa SquareVector;

Main(){
    var int A[0 10], size;

    A[0] = 10;
    A[1] = 12;
    A[2] = 13;
    A[3] = 16;
    A[4] = 11;
    A[5] = 19;
    A[6] = 30;
    A[7] = 25;
    A[8] = 61;
    A[9] = 123;
    A[10] = 1235;

    print("SQUARE DE VECTOR");
    A.print();
    A.square();
    A.print();
}

```

## Cuádruplos generados:

```

Python
0 []
1 [GOTO, '', '', 2]
2 [VERIFY, 3001, 3001, 3002]
3 ['+', 3001, 3003, 2001]
4 ['+', 2001, 3004, 2002]
5 ['=', 3002, '', '*2002']
6 [VERIFY, 3005, 3001, 3002]
7 ['+', 3005, 3003, 2003]
8 ['+', 2003, 3004, 2004]
9 ['=', 3006, '', '*2004']
10 [VERIFY, 3007, 3001, 3002]

```

```

11 ['+', 3007, 3003, 2005]
12 ['+', 2005, 3004, 2006]
13 ['=', 3008, '', '*2006']
14 [VERIFY, 3009, 3001, 3002]
15 ['+', 3009, 3003, 2007]
16 ['+', 2007, 3004, 2008]
17 ['=', 3010, '', '*2008']
18 [VERIFY, 3011, 3001, 3002]
19 ['+', 3011, 3003, 2009]
20 ['+', 2009, 3004, 2010]
21 ['=', 3012, '', '*2010']
22 [VERIFY, 3013, 3001, 3002]
23 ['+', 3013, 3003, 2011]
24 ['+', 2011, 3004, 2012]
25 ['=', 3014, '', '*2012']
26 [VERIFY, 3015, 3001, 3002]
27 ['+', 3015, 3003, 2013]
28 ['+', 2013, 3004, 2014]
29 ['=', 3016, '', '*2014']
30 [VERIFY, 3017, 3001, 3002]
31 ['+', 3017, 3003, 2015]
32 ['+', 2015, 3004, 2016]
33 ['=', 3018, '', '*2016']
34 [VERIFY, 3019, 3001, 3002]
35 ['+', 3019, 3003, 2017]
36 ['+', 2017, 3004, 2018]
37 ['=', 3020, '', '*2018']
38 [VERIFY, 3021, 3001, 3002]
39 ['+', 3021, 3003, 2019]
40 ['+', 2019, 3004, 2020]
41 ['=', 3022, '', '*2020']
42 [VERIFY, 3002, 3001, 3002]
43 ['+', 3002, 3003, 2021]
44 ['+', 2021, 3004, 2022]
45 ['=', 3023, '', '*2022']
46 [PRINT, '', '', 3024]
47 [PRINTMATRIX, 11, 1001, '']
48 [SQUAREVECTOR, 11, 1001, '']
49 [PRINTMATRIX, 11, 1001, '']
50 [END, '', '', '']

```

## Resultado

```
SQUARE DE VECTOR
|| PRINTING MATRIX ||
10
12
13
16
11
19
30
25
61
123
1235
|| END PRINTING MATRIX ||
|| PRINTING MATRIX ||
100
144
169
256
121
361
900
625
3721
15129
1525225
|| END PRINTING MATRIX ||
```

## Bubble Sort

### Código Ejecutado:

```
Python
Programa BubbleSort;

Main(){
    var boolean found;
    var int i, j, aux, A[0 10], size;

    A[0] = 10;
    A[1] = 12;
    A[2] = 13;
    A[3] = 16;
    A[4] = 11;
    A[5] = 19;
    A[6] = 30;
    A[7] = 25;
    A[8] = 61;
    A[9] = 123;
    A[10] = 1235;

    size = 10;
    i = 0;
    j = 1;

    A.print();

    while(i < size){
        j = i + 1;
        while(j < size){
            if(A[i] > A[j]){
                aux = A[i];
                A[i] = A[j];
                A[j] = aux;
            };
            j = j + 1;
        };
        i = i + 1;
    };

    A.print();
}
```

## Cuádruplos generados

Python

```

0 []
1 [GOTO, '', '', 2]
2 [VERIFY, 3001, 3001, 3002]
3 ['+', 3001, 3003, 2001]
4 ['+', 2001, 3004, 2002]
5 ['=', 3002, '', '*2002']
6 [VERIFY, 3005, 3001, 3002]
7 ['+', 3005, 3003, 2003]
8 ['+', 2003, 3004, 2004]
9 ['=', 3006, '', '*2004']
10 [VERIFY, 3007, 3001, 3002]
11 ['+', 3007, 3003, 2005]
12 ['+', 2005, 3004, 2006]
13 ['=', 3008, '', '*2006']
14 [VERIFY, 3009, 3001, 3002]
15 ['+', 3009, 3003, 2007]
16 ['+', 2007, 3004, 2008]
17 ['=', 3010, '', '*2008']
18 [VERIFY, 3011, 3001, 3002]
19 ['+', 3011, 3003, 2009]
20 ['+', 2009, 3004, 2010]
21 ['=', 3012, '', '*2010']
22 [VERIFY, 3013, 3001, 3002]
23 ['+', 3013, 3003, 2011]
24 ['+', 2011, 3004, 2012]
25 ['=', 3014, '', '*2012']
26 [VERIFY, 3015, 3001, 3002]
27 ['+', 3015, 3003, 2013]
28 ['+', 2013, 3004, 2014]
29 ['=', 3016, '', '*2014']
30 [VERIFY, 3017, 3001, 3002]
31 ['+', 3017, 3003, 2015]
32 ['+', 2015, 3004, 2016]
33 ['=', 3018, '', '*2016']
34 [VERIFY, 3019, 3001, 3002]
35 ['+', 3019, 3003, 2017]
36 ['+', 2017, 3004, 2018]
37 ['=', 3020, '', '*2018']
38 [VERIFY, 3021, 3001, 3002]
39 ['+', 3021, 3003, 2019]
40 ['+', 2019, 3004, 2020]
41 ['=', 3022, '', '*2020']

```



```

42 [VERIFY, 3002, 3001, 3002]
43 ['+', 3002, 3003, 2021]
44 ['+', 2021, 3004, 2022]
45 ['=', 3023, '', '*2022']
46 ['=', 3002, '', 1015]
47 ['=', 3001, '', 1001]
48 ['=', 3005, '', 1002]
49 [PRINTMATRIX, 11, 1004, '']
50 ['<', 1001, 1015, 2501]
51 [GOTO, 2501, '', 85]
52 ['+', 1001, 3005, 2023]
53 ['=', 2023, '', 1002]
54 ['<', 1002, 1015, 2502]
55 [GOTO, 2502, '', 82]
56 [VERIFY, 1001, 3001, 3002]
57 ['+', 1001, 3003, 2024]
58 ['+', 2024, 3004, 2025]
59 [VERIFY, 1002, 3001, 3002]
60 ['+', 1002, 3003, 2026]
61 ['+', 2026, 3004, 2027]
62 ['>', '*2025', '*2027', 2503]
63 [GOTO, 2503, '', 79]
64 [VERIFY, 1001, 3001, 3002]
65 ['+', 1001, 3003, 2028]
66 ['+', 2028, 3004, 2029]
67 ['=', '*2029', '', 1003]
68 [VERIFY, 1001, 3001, 3002]
69 ['+', 1001, 3003, 2030]
70 ['+', 2030, 3004, 2031]
71 [VERIFY, 1002, 3001, 3002]
72 ['+', 1002, 3003, 2032]
73 ['+', 2032, 3004, 2033]
74 ['=', '*2033', '', '*2031']
75 [VERIFY, 1002, 3001, 3002]
76 ['+', 1002, 3003, 2034]
77 ['+', 2034, 3004, 2035]
78 ['=', 1003, '', '*2035']
79 ['+', 1002, 3005, 2036]
80 ['=', 2036, '', 1002]
81 [GOTO, '', '', 54]
82 ['+', 1001, 3005, 2037]
83 ['=', 2037, '', 1001]
84 [GOTO, '', '', 50]
85 [PRINTMATRIX, 11, 1004, '']
86 [END, '', '', '']

```

Resultado:

```
running
|| PRINTING MATRIX ||
10
12
13
16
11
19
30
25
61
123
1235
|| END PRINTING MATRIX ||
|| PRINTING MATRIX ||
10
11
12
13
16
19
25
30
61
123
1235
|| END PRINTING MATRIX ||
```

## DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO

Durante el desarrollo de la documentación se documentó adecuadamente los métodos de cada función así como con sus parámetros, lo mismo para sus clases, favor de referirse a este pasado para conocer documentación del código puntual

# Quick Reference

## Instalación:

Descargar repositorio y descomprimir en una carpeta, asegurarse de tener python instalado y asignado en el path de la terminal

- Agregar archivo txt a carpeta de test
- abrir una terminal
- navegar por la terminal a la carpeta donde se encuentra main.py
- correr el comando `python main.py archivo.txt`
- La terminal correrá el archivo.txt, primero desplegará los cuádruplos generados y después le ejecución del programa.
- 

CompiMoi sigue una estructura de programación convencional. Primero se define el nombre del programa, luego variables globales, módulos y al final una Main Function.

- Todos los programas deben de empezar con la palabra reservada 'Programa' seguido del nombre del programa con un punto y coma.
- Después se pueden definir variables globales, las variables siguen la siguiente estructura: **var {tipo} id, id2, id3;** Siendo los tipos posibles: **int, float, boolean, string.**
- Por el momento nuestro programa se vería así

Python

```
Program Test;
var int a,b,c;
var float a,b,c;
```

- Después se pueden o no declara módulos, los módulos contienen la siguiente estructura

Python

```
Function int printVariable(int aux){
    print(aux);
}
```

- De este ejemplo se puede apreciar que las funciones pueden tener un tipo de retorno como las variables, o bien se pueden dejar como **void**, los parámetros también pueden tener tipo, y podemos ver un ejemplo de **print**
- A continuación se pueden declarar uno o más módulos, y al final un Main, que no es nada más que una función con algunos cambios, por ejemplo esta no recibe parámetros ni ocupa declarar un tipo de retorno

Python

```

Main(){
    var int aux;
    aux = 0;
    print(aux);
}

```

- Estatutos no lineales: se cuenta con estructuras como while, y If-Elses, su estructura sería como la común en distintos lenguajes de programación: Aquí unos ejemplos

Python

```

var boolean found;
found = True;
while(found){
    found = False;
};

if(found){
    print("HOLA");
}else{
    print("ADIOS");
};

```

- Leer de terminal

Python

```

var int a;
read(a);

```

- Inicializar array y asignar valor

Python

```

var int A[0 1, 2, 3];
A[1,2] = 10;

```

- Los arrays contienen métodos especiales, contienen el método print() que imprime linealmente la matriz/array y los vectores de una dimensión contienen el método square que elevan al cuadrado cada elemento del vector

Python

```
var int A[0 1, 2, 3];  
A[1,2] = 10;  
A.square();  
A.print();
```

Para más información favor de consultar sección de sintaxis.

## VIDEODEMO

[Link](#)