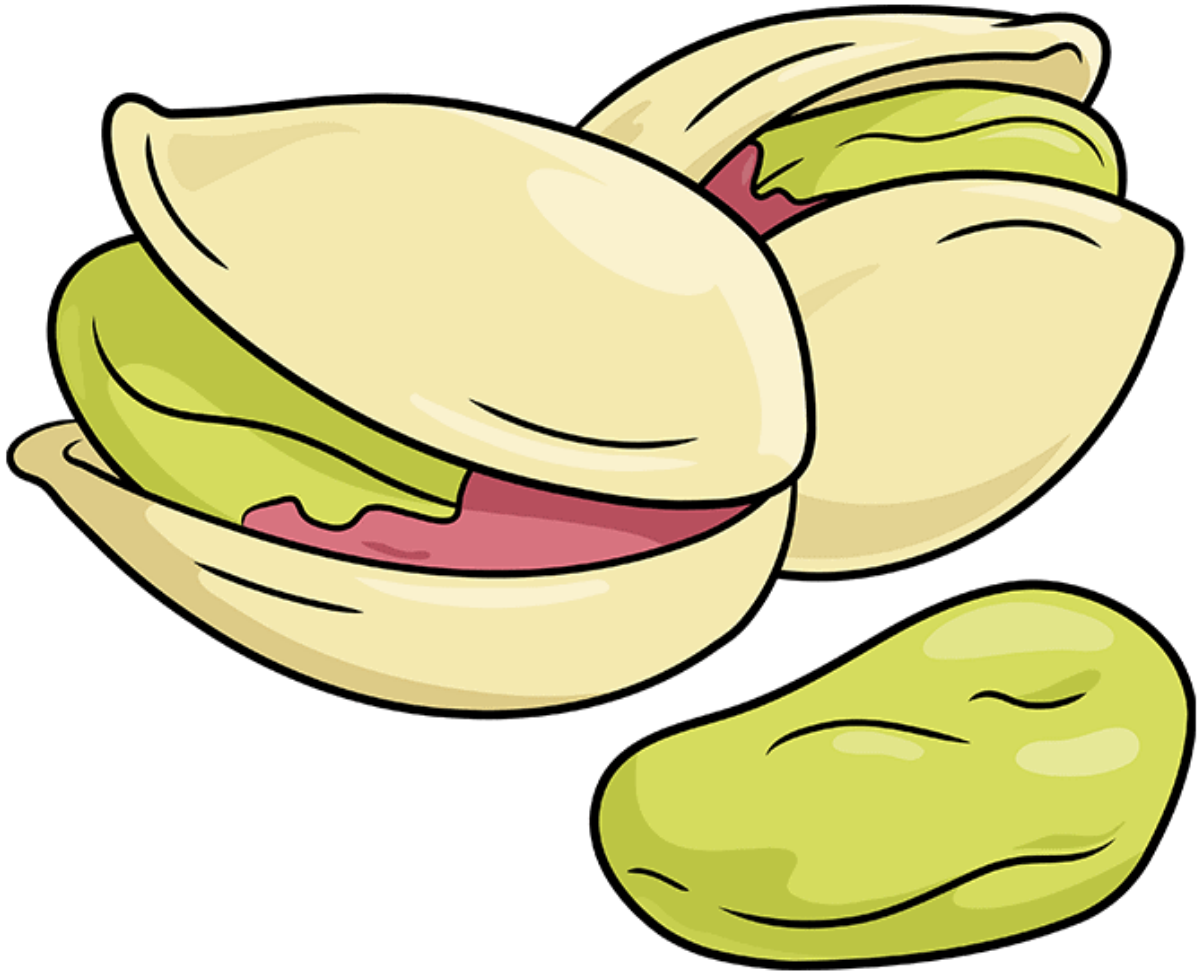


# Pystachio



Pamela Lozano de la Garza

22 de Noviembre del 2022

A01176970

# Índice

<b>Pystachio</b>	<b>1</b>
Índice	2
Aprendizajes	4
Descripción del proyecto	5
Propósito y Alcance del Proyecto.	5
Análisis de Requerimientos y descripción de los principales Test Cases.	5
Proceso del desarrollo	6
Descripción del Lenguaje	9
Nombre	9
Descripción	9
Imprevistos que pueden ocurrir durante ejecución	9
Descripción del Compilador	10
Tecnologías y Librerías	10
Análisis de Léxico	10
Tokens	10
Análisis de Sintaxis	11
Diagrama de Sintaxis	11
Gramática Formal	12
Descripción de Generación de Código Intermedio y Análisis Semántico	14
Códigos de Operación y Direcciones asociadas a los elementos del código	14
Diagramas de Sintaxis y sus Puntos Neurálgicos	16
START	16
ASIGNACIÓN	16
RETURN / BREAK	16
ESCRITURA	17
WHILE	17
DO WHILE	17
FUNCTION	18
FUNCTIONCALL	18
CONDICIÓN	19
FORLOOP	19
ARRAY	19
MATRIX	20
ARRCALL	20
ASIGNACIONARR	20
OPERACIONES ARITMÉTICAS	21
Consideraciones Semánticas	22
Cubo Semántico	22
INT	22
FLOAT	23

BOOL	24
STRING	24
Tipos de Errores	25
Sintaxis	25
Semántica	25
Ejecución	25
Descripción del proceso de Administración de Memoria usado en la compilación.	26
Tabla de Funciones	26
1. Tabla de Variables	26
2. Constantes	26
3. Argumentos	26
4. Contador de Recursos	26
Tabla de Arreglos	27
Cuádruplos	28
Pilas	28
PilaO	28
PType	28
POper	29
IsPointer	29
AssignToArray	29
Descripción de la Máquina Virtual	30
Administración de Memoria	30
Direcciones de Memoria	30
Almacenamiento	31
Pruebas	31
Arithmetic	32
While Loops	33
Fibonacci Cíclico	34
Fibonacci Recursivo	35
Sort (For anidado)	36
Find in Matrix	37
Manual de Usuario	38
1. Instalar Python	38
2. Descargar zip de repositorio	38
3. Abrir zip y localizar ubicación del folder	38
4. Abrir terminal o CMD y mover ubicación al path del folder 'compiladores'.	38
5. Crear un archivo .pyst en la carpeta 'compiladores'.	38
6. Correr run.py en la terminal	38
Video Demostración	38
Ejemplo	38

## Aprendizajes

La principal problemática con la que me enfrenté al principio del proyecto fue la planeación de propuesta y la elaboración de la sintaxis y el léxico; ver los temas de gramática, vectores polacos y cuádruplos con anticipación se volvió abrumador. Una vez que empecé el proyecto todo se fue facilitando, los temas de la clase empezaron a hacer sentido en el proyecto. Sin embargo, la complejidad del proyecto fue incrementando y también el tiempo invertido en él. Balancear el tiempo entre vida personal, trabajo, las demás materias que cursé durante el semestre y este proyecto fue retador. Personalmente, la planeación y organización de tiempo para las entregas fue uno de los aprendizajes más importantes, ya que siendo un proyecto individual la responsabilidad era 100% mía. Otro aprendizaje importante fue durante la fase de semántica, ya que requirió el repaso de múltiples materias que cursé durante la carrera, cómo algoritmos, estructura de datos, organización computacional, sistemas operativos, entre otros. Al terminar y ver los resultados finales, concluí que ha sido de los proyectos más retadores pero también más interesantes de mi trayecto profesional. Finalmente, quiero agradecer la atención y el apoyo de mi directora de carrera Elda Quiroga durante el desarrollo de este proyecto.

*Pamela Lozano De la Garza*

# Descripción del proyecto

## Propósito y Alcance del Proyecto.

El proyecto consiste en diseñar e implementar un lenguaje de programación en nuestro lenguaje de preferencia. El texto del lenguaje se interpretará como una colección de tokens especificados por las reglas de expresión regular y consecuentemente reemplazar la secuencia de tokens por la sintaxis del lenguaje que se ha especificado en forma de gramática libre de contexto. Durante el análisis de sintaxis también se generará el código intermedio con el análisis de semántica y nuestra propia máquina virtual interpretará la representación intermedia.

El alcance del proyecto incluye el análisis de operaciones aritméticas y comparativas, uso de strings, flujos de condiciones, flujos de ciclos (for y while), ambientes anidados (globales y locales), módulos autónomos reutilizables (funciones), arreglos y matrices.

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

Se requiere la ejecución de lo siguiente:

- Operaciones aritméticas complejas y de asignación.
- Expresiones comparativas.
- Flujos de condiciones.
- Flujos de ciclos For y While (anidados)
- Gestión de memoria en ambientes anidados
- Módulos autónomos reutilizables (funciones)
- Gestión de memoria local, argumentos y valores de retorno de las funciones (recursividad).
- Asignación de valores a arreglos.
- Recopilación de los valores de arreglos.
- Asignación de valores a matrices.
- Recopilación de los valores de matrices.
- Procesamiento de input del usuario
- Output a la terminal
- Impresión de Errores

## Proceso del desarrollo

### [Link de repositorio](#)

#### **Commits on Nov 19, 2022**

Add input  
Find in matrix test  
Test find  
Sort test  
Calculation for matrix direction  
Test fib cíclico  
Test fibonacci recursivo

#### **Commits on Nov 18, 2022**

Operations with array pointer  
**Check if there is a pointer before** appending each quad  
Fix reassign in quad **when** its pointer  
Arrays pointer **as** expresión  
Quad **for** a[i]=x adds a[i] **after** x **to** quads  
Assign **to** arrays máquina **virtual**  
Assign **to** arrays  
Run **file**

#### **Commits on Nov 17, 2022**

Fix recursividad  
Máquina **virtual** recursividad  
Break keyword **to** end funcs  
**Add RETURN** quad **to** endfunc  
Máquina **virtual** funcs  
Máquina **virtual** **for**  
Máquina **virtual** gotos

#### **Commits on Nov 16, 2022**

**Change == to !=**  
Find vars dir **function search from current scope**  
Correr máquina desde sintaxis  
**Virtual machine operations +-\* / and** assign  
Creación memoria en maquina **virtual**  
Fix print  
Fix strings **and** bools

#### **Commits on Nov 14, 2022**

Documentación avance 7  
Fix syntax **with ]**  
Arrays quads

**Commits on Nov 12, 2022**

Calculate array offset

Array syntax

Error handling, quit reading in Exception

Operations with functions

**Commits on Nov 10, 2022**

Update avance 6 tests

Return values and save in global

**Commits on Nov 7, 2022**

Assign return to temp

**Commits on Nov 4, 2022**

Funciones test

Remove varCheck

Add initial goto main

Function quadruples

Add initial address to func table

**Commits on Nov 3, 2022**

Reset temps and locals

**Commits on Nov 2, 2022**

Syntax for function calls

Add operation codes

Un poco de documentación

Check expression return type match bool (if, while)

**Commits on Nov 1, 2022**

Declare vars in function

Declare function before block

Print quads with names

Constants in their own table

**Commits on Oct 31, 2022**

Use directions in quads

Accept and save constants in global scope

Add directions object and to vars table

**Commits on Oct 29, 2022**

For loops quads

**Commits on Oct 28, 2022**

Add do and while test

**Do loop** quads  
**Quads for while**

**Commits on Oct 27, 2022**  
**Quads for else**

**Commits on Oct 19, 2022**  
Expresiones if y operaciones con comparación

**Commits on Oct 18, 2022**  
**Exp** aritméticas y asignación  
Aritmetic quads (falta fondo negro)  
**Add**, subs, equals quads

**Commits on Oct 17, 2022**  
Readme avance 2

**Commits on Oct 11, 2022**  
**Semantics for storing variables**  
**Variables table and scope control**

**Commits on Oct 10, 2022**  
Avance #0 documentación  
**Add semantic cube**

**Commits on Nov 22, 2022**  
Fix array types



# Descripción del Lenguaje

## Nombre

El nombre Pystachio viene del lenguaje base 'Python' y mi nuez favorita de la familia de los anacardos.

## Descripción

Pystachio un lenguaje de programación de "alto nivel", ya que proporciona una fuerte abstracción de la arquitectura que usa la computadora (1 y 0) y mantiene una estructura similar a la de C++. Sin embargo al utilizar python como intermediario reduce el rendimiento y la eficiencia de memoria. Este lenguaje solo cubre los requerimientos básicos de un lenguaje de programación (variables, funciones, ciclos, condicionales, arreglos, matrices, input y output).

## Imprevistos que pueden ocurrir durante ejecución

- Utilizar break en una función void
  - La palabra *break* funciona como un *return* en Pystachio. Si se utiliza en una función *void* el error no es atrapado por el compilador y aparece el siguiente error:

```
if(self.directions[scope][varType][0] >
self.directions[scope][varType][2]): KeyError: None
```
- Solo funcionan arreglos y matrices de tipo INT y FLOAT
  - Si creamos un arreglo otro tipo, por ejemplo STRING, el compilador toma INT como el tipo default del arreglo, por lo que al asignar un string a un apuntador aparece el siguiente error:

```
ValueError: ('Types mismatch', 'string', 'int', '=')
```
- Confusión de operador != por !=
  - Durante la implementación surgieron problemas usando '==' como el token para la comparación de dos expresiones. Para simplificar la solución cambie el token de == por !=. Esto puede ser confuso ya que en la mayoría de los lenguajes utilizamos != como 'No igual'.

# Descripción del Compilador

## Tecnologías y Librerías

Pystachio es un proyecto escrito con Python, utilicé la librería [PLY](#) para crear el léxico y la sintaxis del compilador. Para utilizar el proyecto, es necesario instalar Python versión 3+, instalar un editor de texto y tener una línea de comando o terminal disponible.

## Análisis de Léxico

### Tokens

START-> start	VOID-> void	RPAREN-> )
VAR-> var	STRING->string	SEMICOLON-> ;
FUNCTION-> function	FLOAT->float	COLON-> :
SWITCH-> switch	INT->int	COMA-> ,
FOR-> for	BOOL->bool	EQUAL-> =
RETURN-> return	VOID->void	MORETHAN-> >
BREAK-> break	LCURLY-> {	LESSTHAN-> <
PRINT-> print	RCURLY-> }	NOTEQUAL-> !=
IF-> if	LBRACKET-> [	ISEQUAL-> ==
ELSE-> else	RBRACKET-> ]	PLUS-> +
WHILE-> while	OR-> !!	MINUS-> -
INPUT-> input	AND-> &&	TIMES-> *
	LPAREN-> (	DIVIDE-> /

**ID** -> [a-zA-Z][a-zA-Z\_0-9]\*

**STRING** -> "[^"]\*"

**FLOAT** -> [0-9]+\.[0-9]+

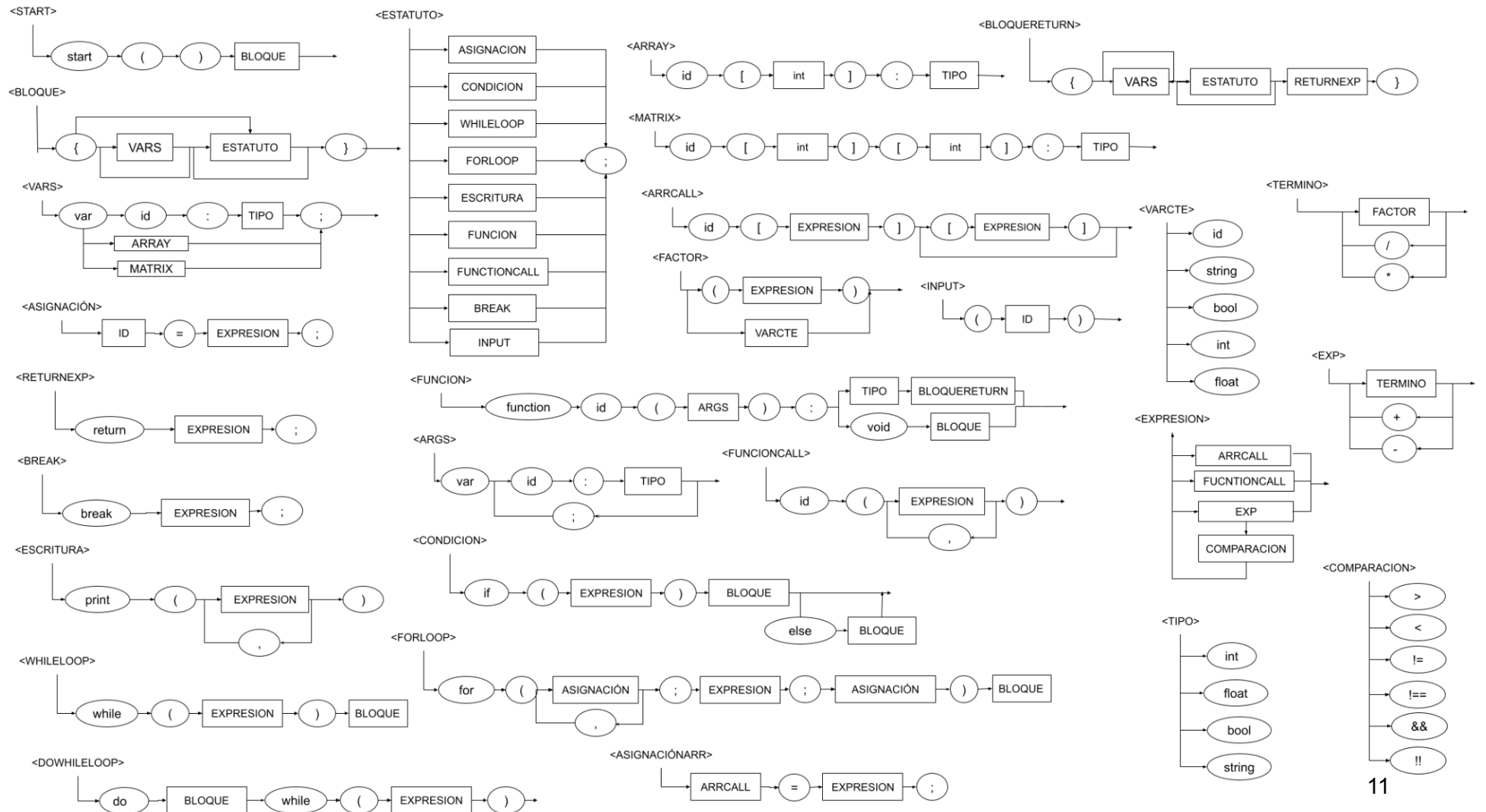
**INT** -> [0-9]+

**BOOL** -> True|False

**NEWLINE** -> \n+

## Análisis de Sintaxis

## Diagrama de Sintaxis



## Gramática Formal

```
programa -> START LPAREN RPAREN bloque
declaracion -> vars | epsilon
vars -> VAR var | vars vars
var -> ID COLON tipo SEMICOLON
      | array SEMICOLON
      | matrix SEMICOLON
vardef -> ID
tipo -> INT
      | FLOAT
      | BOOL
      | STRING
bloque -> LCURLY declaracion estatutoExp RCURLY
bloqueReturn -> LCURLY declaracion estatutoExp returnexp RCURLY
estatutoExp -> estatuto SEMICOLON | estatutoExp estatutoExp
estatuto -> asignacion
          | asignacionArr
          | condicion
          | whileLoop
          | doWhile
          | forLoop
          | escritura
          | funcion
          | functionCall
          | break
          | input
break -> BREAK expresion
returnexp -> RETURN expresion SEMICOLON
asignacion -> ID EQUAL expresion
escritura -> PRINT LPAREN escrito RPAREN
escrito -> expresion | expresion COMA escrito
expresion -> exp
          | condition
          | functionCall
          | arr
condition -> exp comparacion expresion
comparacion -> LESSTHAN
              | MORETHAN
              | ISEQUAL
              | NOTEQUAL
              | AND
              | OR
```

```

doWhile -> DO bloque WHILE LPAREN expresion RPAREN
whileLoop -> WHILE startCondition expresion endCondition bloque
forLoop -> FOR LPAREN argumentos startCondition expresion endCondition
asignacion RPAREN bloque
argumentos -> args | epsilon
args -> asignacion | args COMA args
condicion -> IF LPAREN expresion endCondition bloque condicionelse
startCondition -> LPAREN | SEMICOLON
endCondition -> RPAREN | SEMICOLON
condicionelse -> ELSE bloque | epsilon
funcion -> FUNCTION ID LPAREN declaracion RPAREN COLON tiposreturn
tiposreturn -> tiposFuncion bloqueReturn | VOID bloque
tiposFuncion -> INT
                | FLOAT
                | BOOL
                | STRING
functionCall -> ID LPAREN funcArgs RPAREN
funcArgs -> expresion | expresion COMA funcArgs | epsilon
exp -> termino | termino signo exp
signo -> PLUS | MINUS
termino -> factor | factor operacion termino
operacion -> TIMES
                | DIVIDE
                | DIFF
                | EXP
factor -> varcte | LPAREN expresion RPAREN | functionCall
epsilon -> ε
array -> ID LBRACKET INT RBRACKET COLON tipo
matrix -> ID LBRACKET INT RBRACKET LBRACKET INT RBRACKET COLON tipo
assignArr -> arr EQUAL expresion
arr -> ID LBRACKET expresion abracket matrix
        | ID LBRACKET expresion abracket epsilon
matrixCall -> LBRACKET expresion mbracket
arrCall -> RBRACKET
mbracket -> RBRACKET
input -> INPUT LPAREN ID RPAREN
varcte -> ID
                | INT
                | FLOAT
                | BOOL
                | STRING
                | matrix
                | arr

```

## Descripción de Generación de Código Intermedio y Análisis Semántico

### Códigos de Operación y Direcciones asociadas a los elementos del código

```
'operationCodes': {
  '+':1,
  '-':2,
  '*':3,
  '/':4,
  '=':5,
  '>':6,
  '<':7,
  '!=':8,
  '!==':9,
  '%':10,
  '!!':11,
  '&&':12,
  '^':13,
  'end':14
},
'global': {
  'int' : [0, 15, 1000],
  'float' : [0, 1001, 2000],
  'bool' : [0, 2001, 3000],
  'string' : [0, 3001, 4000],
},
'local': {
  'int' : [0, 4001, 5000],
  'float' : [0, 5001, 6000],
  'bool' : [0, 6001, 7000],
  'string' : [0, 7001, 8000],
},
'const': {
  'int' : [0, 8001, 9000],
  'float' : [0, 9001, 10000],
  'bool' : [0, 10001, 11000],
  'string' : [0, 11001, 12000],
},
'temp': {
  'int' : [0, 12001, 13000],
  'float' : [0, 13001, 14000],
  'bool' : [0, 14001, 15000],
  'string' : [0, 15001, 16000]}
```

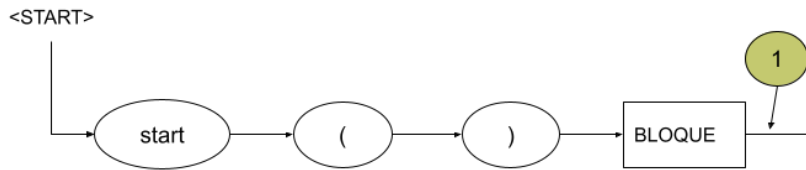
Estas direcciones pertenecen a la clase [DirectionsControl](#) y son utilizadas por la clase [VariablesControl](#), donde guardamos esta dirección y la utilizamos en la tabla de funciones para asociar la dirección con el nombre, el scope, constantes, contador de recursos, dirección inicial, etc. También la utilizamos para borrar la memoria local cuando termina una función. Podemos encontrar la estructura de la tabla de funciones en la [sección 5](#).

Por ejemplo, al hacer **PilaO.pop()** lo que obtenemos es el nombre de la variable, sin embargo, necesitamos la dirección para crear el cuádruplo. Entonces podemos llamar la siguiente function en VariablesControl:

```
def find_vars_dir(self, name):
    current_scope = self.current_scope
    while current_scope != '#': #Sube hasta global
    try:
        indx = self.variables_table[current_scope]['vars_table'].index(name)
        #La dirección que guardamos con DirectionsControl
        varDir=self.variables_table[current_scope]['vars_dir'][indx]
        return varDir
    except:
        current_scope = self.variables_table[current_scope]['scope']
    try:
        indx=self.constants['vars_table'].index(name)
        varDir=self.constants['vars_dir'][indx]
        return varDir
    except:
        return None
```

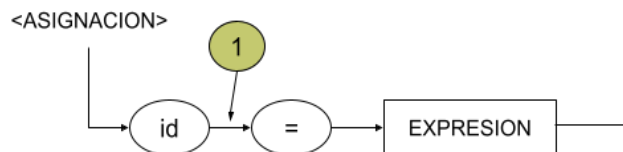
## Diagramas de Sintaxis y sus Puntos Neurálgicos

### START



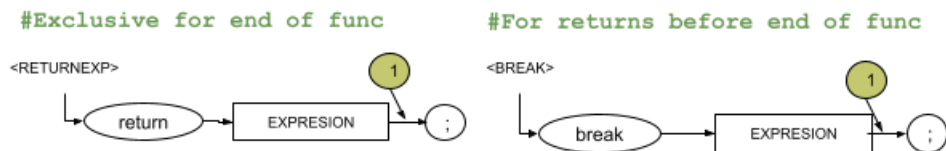
1. Quad('end', None, None, None)  
quads.append(Quad)  
Create ovejota.json = {  
    "quads":self.quads,  
    "function\_table": funcTable,  
    "global":funcTable['global'],  
    "constants": variables\_control.getConstants(),  
    "args\_table": variables\_control.get\_args\_table()  
}

### ASIGNACIÓN



1. IF id exists: ValueError("Variable is already declared")  
ELSE:  
    pTypes.append(idType)  
    pilaO.append(id)  
    pOper.append("=")  
    IF pOper.top == "=": Quad(pOper.pop, pilaO.pop, pilaO.pop, temp)

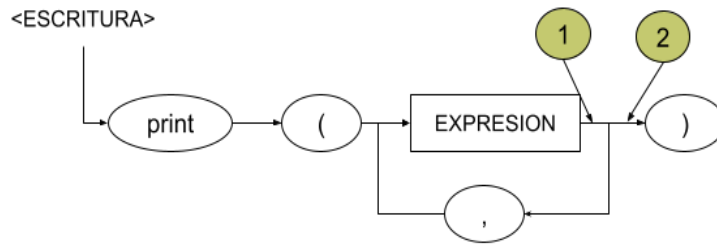
### RETURN / BREAK



1. current\_scope = global  
pilaO.push(add\_var(func\_name, expected\_return\_type))  
pOper.push("=")  
IF pOper.top == "=": Quad(pOper.pop, pilaO.pop, pilaO.pop, temp)  
Quad("RETURN", none, none, none)

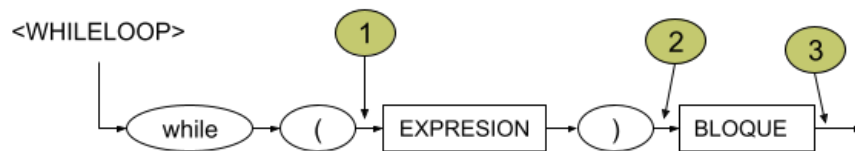


## ESCRITURA



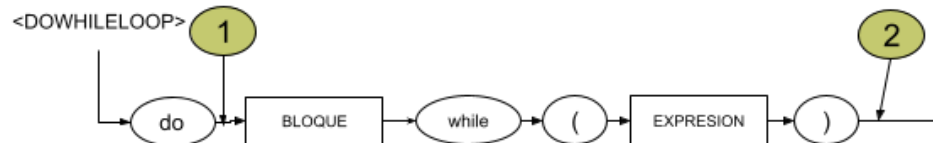
1. pOper.push("PRINT")
2. IF print == "PRINT": Quad("PRINT", None, None, pilaO.pop())

## WHILE



1. pSaltos.push(counter)
2. Quad("GOTOF", pilaO.pop(), None, None)  
IF self.pTypes.pop() != 'bool': raise ValueError("IF, WHILE and FOR blocks are conditional")  
pSaltos.push(counter)
3. Quads[pSaltos.pop()][-1] = counter+2 #Skip next goto quad
4. Quad("GOTO", None, None, pSaltos.pop())

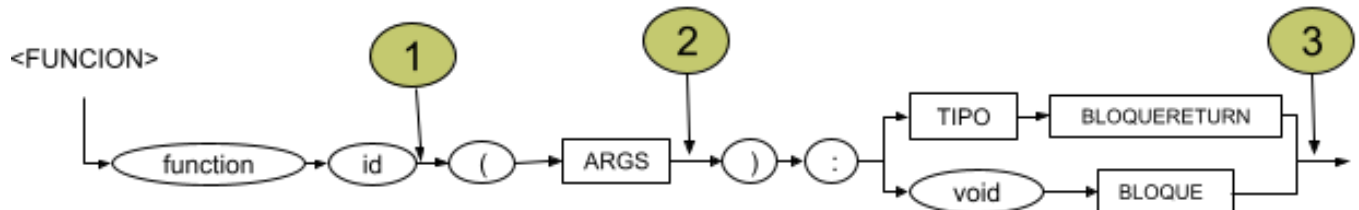
## DO WHILE



1. pSaltos.push(counter)
2. Quad("GOTOT", pilaO.pop(), None, None)  
IF self.pTypes.pop() != 'bool': raise ValueError("IF, WHILE and FOR blocks are conditional")  
pSaltos.push(counter)
3. Quads[pSaltos.pop()][-1] = pSaltos.pop()

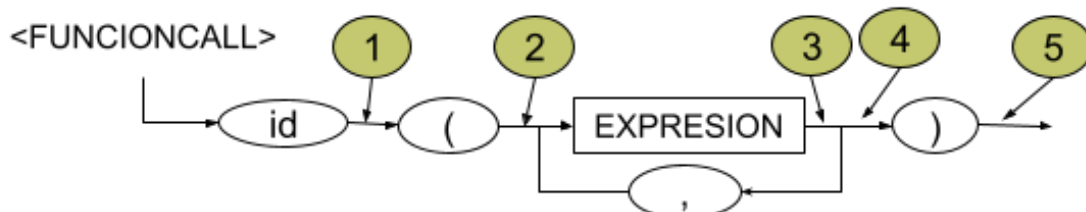
## FUNCTION

En la función utilizamos la sintaxis para verificar si el bloque tiene un return cuando el *return\_type* no es *void*. Por esta razón, se volvió complicado agregar *return* antes del final del bloque. Como solución implementé el token *break*, que tiene el mismo efecto que el return con la diferencia de que puede ser utilizado como estatuto en cualquier parte del bloque.



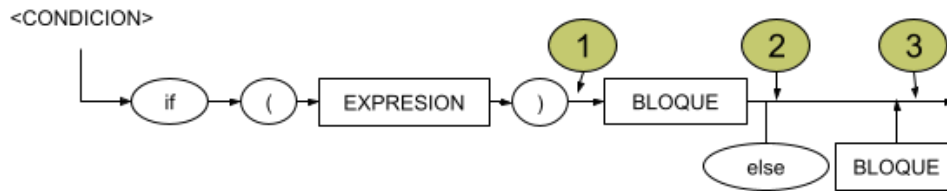
1. IF id exists: ValueError("Variable is already declared")  
ELSE: variables\_control.add\_func(id)
2. variables\_table[func\_name]['initial\_address'] = counter  
FOR i IN ARGS: variables\_table[func\_name]['args'].push(getType(i))
3. variables\_table[func\_name]['return'] = return\_type  
IF func\_name == 'main': self.quads[0][-1] = initial\_address  
Quad(ENDPROC, None, None, None)  
directions\_control.resetLocalCounter()  
current\_scope = global

## FUNCTIONCALL



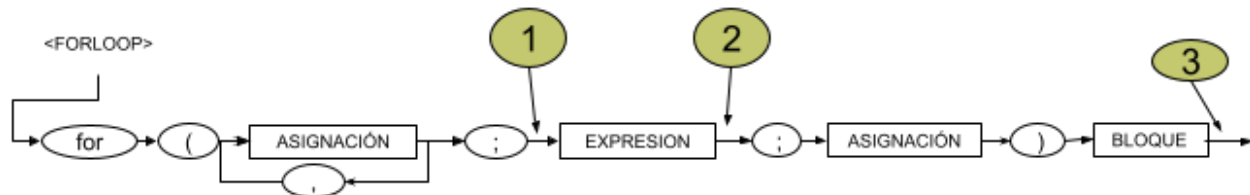
1. IF id not exist: ValueError("Function is not declared")  
Quad("ERA", name, None, None)
2. pOper.push('(')
3. variables\_table[func\_name]['initial\_address'] = counter  
IF pTypes.pop == expectedType: Quad('PARAMETER', pilaO.pop(), #k\_args), k\_args++  
ELSE: raise ValueError('Expected {op} of type {expectedType}')
4. IF pOper.top == '(': pOper.pop() && resolve expresions
5. GOSUB(func\_name, initial\_address, None, None)  
Quad(=, func\_name, None, temp#counter)

## CONDICIÓN



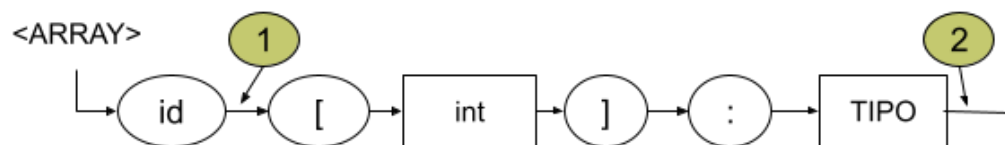
1. Quad("GOTO", pilaO.pop(), None, None)  
IF self.pTypes.pop() != 'bool': raise ValueError("IF, WHILE and FOR blocks are conditional")  
pSaltos.push(counter)
2. Quads[pSaltos.pop()][-1] = counter+1
3. Quad("GOTO", None, None, None)  
pSaltos.push(counter)
4. Quads[pSaltos.pop()][-1] = counter

## FORLOOP



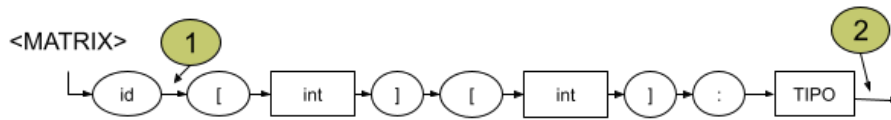
1. pSaltos.push(counter)
2. Quad("GOTO", pilaO.pop(), None, None)  
IF self.pTypes.pop() != 'bool': raise ValueError("IF, WHILE and FOR blocks are conditional")  
pSaltos.push(counter)
3. Quads[pSaltos.pop()][-1] = counter+2 #Skip next goto quad  
Quad("GOTO", None, None, pSaltos.pop())

## ARRAY



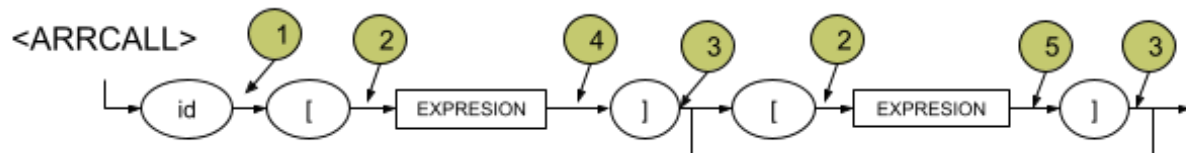
1. IF id exists: ValueError("Variable is already declared")  
ELSE: declaring\_array.push(id)
2. variables\_table[func\_name]['vars\_dir'] = initial\_address  
variables\_table[func\_name][arrays].push({  
    'name': name,  
    'rows': rows,  
    'cols': cols,  
    'type': varType,  
    'initial\_address': varDir  
})  
variables\_table[func\_name]['resource\_count'][varType]+=(rows\*cols)

## MATRIX



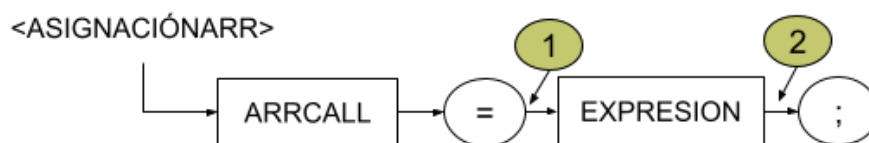
1. IF id exists: ValueError("Variable is already declared")  
ELSE: declaring\_array.push(id)
2. variables\_table[func\_name]['vars\_dir'] = initial\_address  
variables\_table[func\_name]['arrays'].push({  
  'name': name,  
  'rows': rows,  
  'cols': cols,  
  'type': varType,  
  'initial\_address': varDir  
})  
variables\_table[func\_name]['resource\_count']['varType'] += (rows \* cols)

## ARRCALL



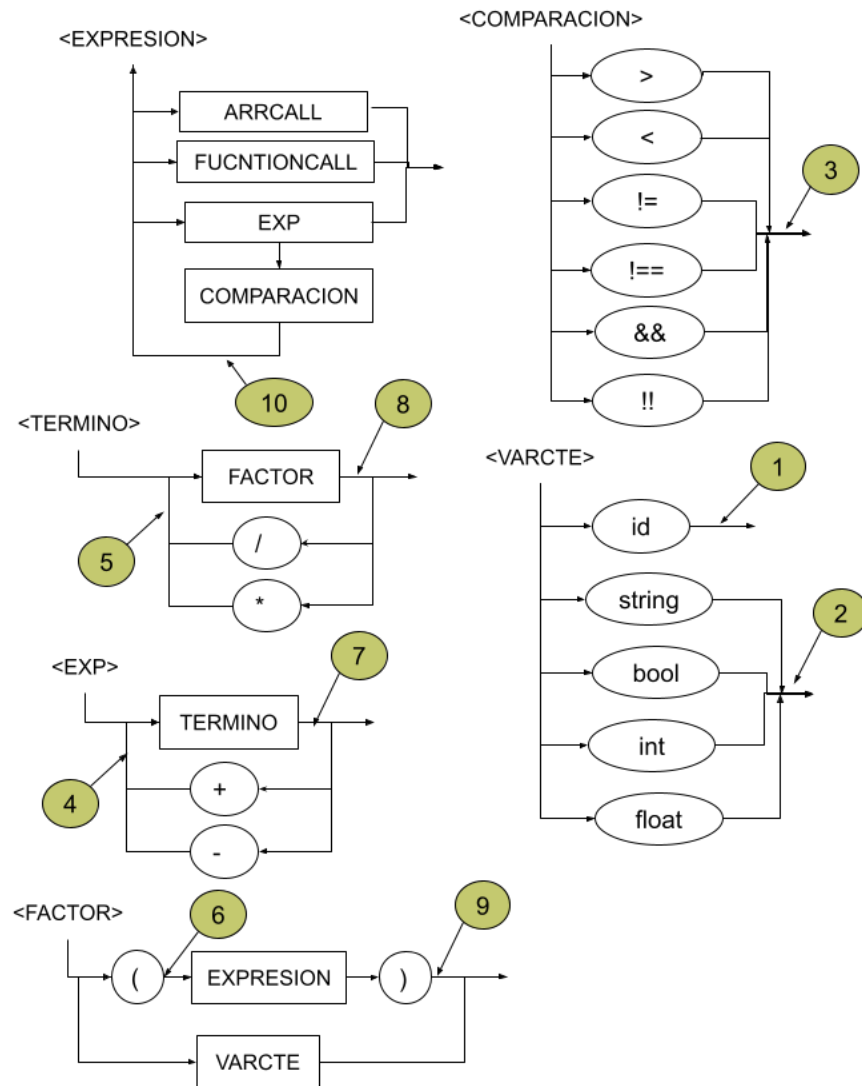
1. IF id not exists: ValueError("Variable is NOT declared")
2. pOper.push('(')
3. IF pOper.top == '(': pOper.pop() && resolve expressions (\*, /, +, =, ><, etc.)
4. IF PType.pop != int: raise ValueError("Array's index must be integers")  
Quad('VER', pilaO.pop, 0, NumRows)  
pilaO.push(variables\_table[func\_name]['vars\_dir'][indx])  
pOper.push('+')
5. IF pOper.top == '+': resolve # Base(arr) + s1
- IF PType.pop != int: raise ValueError("Array's index must be integers")  
Quad('VER', pilaO.pop, 0, NumCols)  
pilaO.push(NumRows)  
pOper.push('\*')
- IF pOper.top == '\*': resolve # Rows \* s2  
pOper.push('+')
- IF pOper.top == '+': resolve # Base(arr) + s1 + Rows \* s2

## ASIGNACIONARR



1. assignToArray.append(self.pilaO.pop())
2. IF pOper.top() == '=' && assignToArray: pilaO.append(assignToArray.pop())

## OPERACIONES ARITMÉTICAS



1. PilaO.push(id)  
PTypes.push(varType)
2. variables\_control.addConst(p[1], var\_type)
3. POper.push(<, >, !=, !=, &&, !!)
4. POper.push(+ or -)
5. POper.push(\* or /)
6. POper.push('(') **#Fondo Negro**
7. IF 0<len(self.pOper) && (pOper[-1]=="+" or pOper[-1]=="-"):
  - temp = nextTempVar()
  - Quad(pilaO.pop(), pilaO.pop(), PTypes.pop(), PTypes.pop(), temp)
  - if result is pointer:
  - isPointer.append(temp)
  - FOR i IN Quad:
  - IF Quad[i] IN isPointer: Quad[i] = f'(Quad[i])'
  - quads.append(Quad)
  - pilaO.append(temp)
  - pTypes.append(typeTemp)
  - tempCounter+=1
8. IF 0<len(self.pOper) && (pOper[-1]=="\*" or pOper[-1]=="/"):
  - = #5 with \*, /
9. IF 0<len(self.pOper) && (pOper[-1]=="("):
  - = #5 with (
10. IF 0<len(self.pOper) && (pOper[-1]=="&&" or pOper[-1]=="!!" or pOper[-1]==">" or pOper[-1]=="<" or pOper[-1]=="!=" or pOper[-1]=="!="):
  - = #5 with <, >, !=, !=, &&, !!

## Consideraciones Semánticas

### Cubo Semántico

#### INT

LEFT	RIGHT	OPERATOR	RESULT
INT	INT	+	INT
INT	INT	-	INT
INT	INT	*	INT
INT	INT	/	INT
INT	INT	=	INT
INT	INT	<	BOOL
INT	INT	>	BOOL
INT	INT	!!	BOOL
INT	INT	&&	BOOL
INT	INT	!=	BOOL
INT	INT	!==	BOOL

LEFT	RIGHT	OPERATOR	RESULT
INT	FLOAT	+	FLOAT
INT	FLOAT	-	FLOAT
INT	FLOAT	*	FLOAT
INT	FLOAT	/	FLOAT
INT	FLOAT	=	FLOAT
INT	FLOAT	<	BOOL
INT	FLOAT	>	BOOL
INT	FLOAT	!!	BOOL
INT	FLOAT	&&	BOOL

INT	FLOAT	!=	BOOL
INT	FLOAT	!==	BOOL

LEFT	RIGHT	OPERATOR	RESULT
INT	BOOL	!=	BOOL
INT	BOOL	!==	BOOL

**FLOAT**

LEFT	RIGHT	OPERATOR	RESULT
FLOAT	INT	+	FLOAT
FLOAT	INT	-	FLOAT
FLOAT	INT	*	FLOAT
FLOAT	INT	/	FLOAT
FLOAT	INT	=	FLOAT
FLOAT	INT	<	BOOL
FLOAT	INT	>	BOOL
FLOAT	INT	!!	BOOL
FLOAT	INT	&&	BOOL
FLOAT	INT	!=	BOOL
FLOAT	INT	!==	BOOL

LEFT	RIGHT	OPERATOR	RESULT
FLOAT	FLOAT	+	FLOAT
FLOAT	FLOAT	-	FLOAT
FLOAT	FLOAT	*	FLOAT
FLOAT	FLOAT	/	FLOAT
FLOAT	FLOAT	=	FLOAT
FLOAT	FLOAT	<	BOOL

FLOAT	FLOAT	>	BOOL
FLOAT	FLOAT	!!	BOOL
FLOAT	FLOAT	&&	BOOL
FLOAT	FLOAT	!=	BOOL
FLOAT	FLOAT	!==	BOOL

LEFT	RIGHT	OPERATOR	RESULT
FLOAT	BOOL	!=	BOOL
FLOAT	BOOL	!==	BOOL

**BOOL**

LEFT	RIGHT	OPERATOR	RESULT
BOOL	BOOL	!!	BOOL
BOOL	BOOL	&&	BOOL
BOOL	BOOL	!=	BOOL
BOOL	BOOL	!==	BOOL

**STRING**

LEFT	RIGHT	OPERATOR	RESULT
STRING	BOOL	!=	BOOL
STRING	BOOL	!==	BOOL

LEFT	RIGHT	OPERATOR	RESULT
STRING	STRING	=	STRING
STRING	STRING	!==	STRING
STRING	STRING	!=	STRING



## Tipos de Errores

**Sintaxis**

```
try: parse except: raise EOFError(f"Syntax error in input! {error}")
```

**Semántica**

```
IF id exists: ValueError('Variable is already declared')
```

```
IF NOT id exists: ValueError('Variable is not declared')
```

```
IF func_name exists: raise ValueError(f'Function {p[1]} is declared twice.')
```

```
IF NOT func_name exists: raise ValueError(f'Function not declared {p[1]}')
```

```
IF self.pTypes.pop() != 'bool': raise ValueError("IF, WHILE and FOR blocks are conditional")
```

```
IF func_arg[i] != expected_type: ValueError(f'Expected {op} of type {expectedType}')
```

```
IF(self.directions[scope][varType][counter] > self.directions[scope][varType][end]): raise MemoryError("Ran out of memory")
```

```
IF NOT Quad(VER, s1, linf, lsup): raise ValueError(f'Index out of range: {value1} in [{value2+1}, {value3}]')
```

```
IF (s1 != 'int'): raise ValueError("Array's index must be integers")
```

```
IF (input != input_var_type): raise ValueError("Must enter same type as variable")
```

**Ejecución**

```
IF len(self.quads) > 10000:
    raise OverflowError('Stack overflow: too many calls')
```

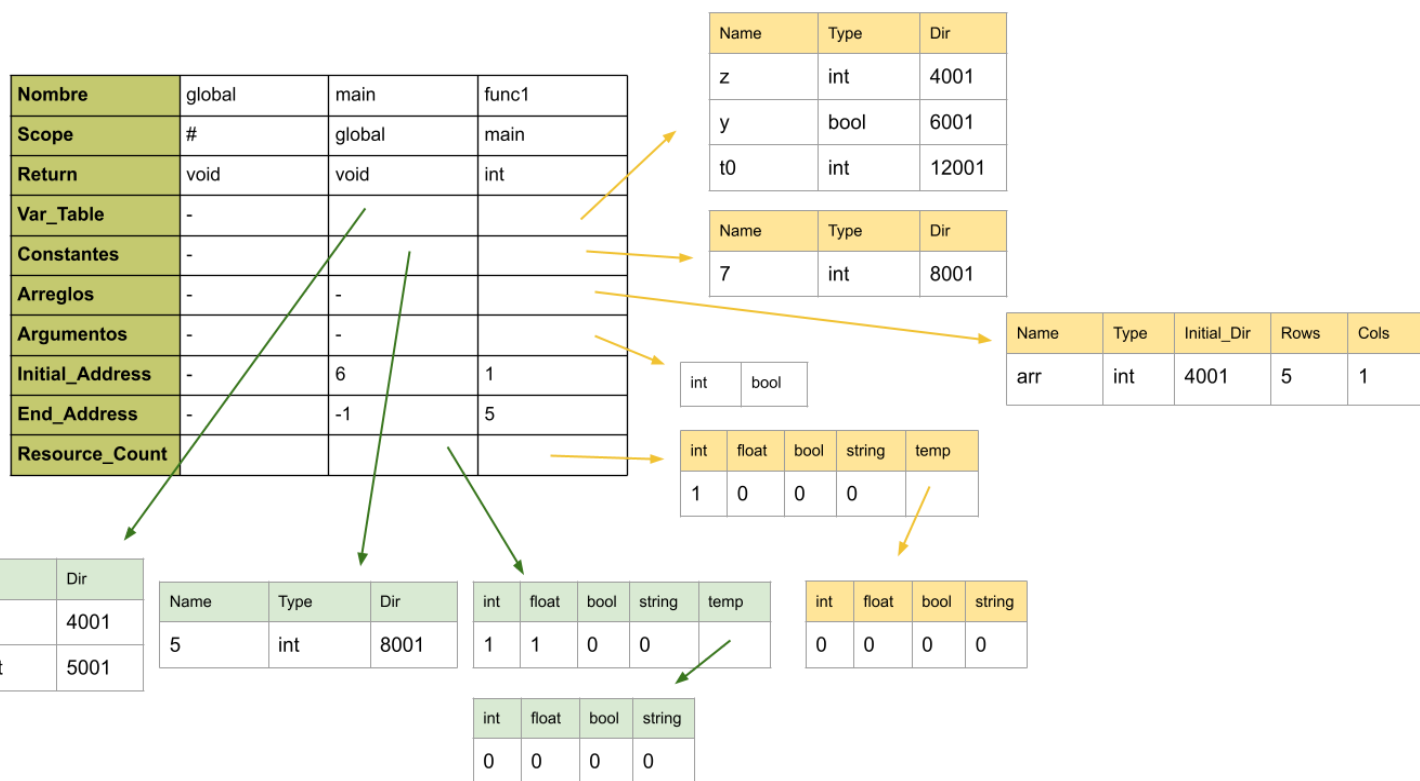
```
IF quad IS arithmetic and NOT value1 OR value2:
    raise ValueError("Added operators should have values")
    raise ValueError("Subtracted operators should have values")
    raise ValueError("Multiplied operators should have values")
    raise ValueError("Divided operators should have values")
    raise ValueError("Compared operators should have values")
    raise ValueError("Exp operators should have values")
    raise ValueError("Compared operators should have values")
```

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

### Tabla de Funciones

Dentro de la tabla de funciones se encuentran los siguientes elementos:

1. Tabla de Variables
2. Constantes
3. Argumentos
4. Contador de Recursos



A pesar de que hay ciertos datos repetitivos, cómo la dirección de memoria (que indica el tipo de dato) y las columnas de 'Tipo', decidí utilizar esta estructura para ahorrar llamadas a funciones de búsqueda. De esta manera solo se busca la dirección una vez ingresando el tipo de dato y el scope (global, constante, local) en la [tabla de direcciones](#).

Una propiedad adicional a la tabla de lo visto en clase es el *End\_Address*, esta se utiliza para recorrer el *instruction\_pointer* cuando se procesa cuádruplo RETURN, que puede ser generado por el token *return* o el token *break*.

## Tabla de Arreglos

Pystachio solamente maneja dos tipos de estructuras no-atómicas, arreglos y matrices. La fórmula y la estructura de datos vista en clase funciona para  $n$  dimensiones, sin embargo, utiliza un proceso más extenso.

Mi propuesta para las direcciones de arreglos y estructuras está compuesta de la siguiente manera:

**Arreglo**

4001	4002	4003
------	------	------

**Matriz**

4001	4004	4007
4002	4005	4008
4003	4006	4009

En base a esta propuesta, podemos utilizar las siguientes fórmulas para resolver el acceso a un arreglo:

$$Address(id[s1]) = BaseA(id) + s1$$

$$Address(id[s1][s2]) = BaseA(id) + s1 + NumRows * s2$$

## Cuádruplos

Se inicializa el cuádruplo en la clase [Cuadruple](#) como un objeto con cuatro llaves: operator, left, right, and result. Después de asignar los valores este cuádruplo se regresa como un arreglo con 4 elementos. También existe una función para crear cada tipo de cuádruplo (GOTOS, RETURN, ERA, etc.). Fue creado de esta manera para que fuera más entendible lo que significa cada lugar del arreglo cuando está dentro de la lista de Quads.

La lista de Quads se encuentra en la clase de [Semántica](#) donde se guardan las direcciones de variables y se utiliza como pila. Al finalizar la compilación, se guarda en *ovejota.json* cómo *quads*.

### Ejemplo:

```
pamela@lozano:YY2VPQ6GXC$ python3 run.py aritmetico
Quads:
0 [goto, 6]
1 [*, z, 5, t0]
2 [=, t0, None, res]
3 [=, res, None, jeje]
4 [RETURN]
5 [ENDPROC]
6 [>, 222, 333, t0]
7 [=, t0, None, ja]
8 [=, 3, None, z]
9 [=, 8, None, res]
10 [*, z, res, t0]
11 [=, t0, None, c]
12 [*, z, res, t2]
13 [*, t2, 2, t3]
14 [/ , t3, 3, t4]
15 [*, 5, 6, t5]
16 [&&, True, 10002, t6]
17 [!=, pamela, pamela, t7]
18 [PRINT, ja]
19 [PRINT, t7]
20 [PRINT, t6]
21 [PRINT, t5]
22 [PRINT, t4]
23 [end, None, None, None]
Quads:
0 ['goto', None, None, 6]
1 [3, 4001, 8001, 12001]
2 [5, 12001, None, 4002]
3 [5, 4002, None, 16]
4 ['RETURN', None, None, None]
5 ['ENDPROC', None, None, None]
6 [6, 8002, 8003, 14001]
7 [5, 14001, None, 6001]
8 [5, 8004, None, 4001]
9 [5, 8005, None, 4002]
10 [3, 4001, 4002, 12001]
11 [5, 12001, None, 15]
12 [3, 4001, 4002, 12002]
13 [3, 12002, 8006, 12003]
14 [4, 12003, 8004, 12004]
15 [3, 8001, 8007, 12005]
16 [12, 10001, 10002, 14002]
17 [9, 11001, 11001, 15001]
18 ['PRINT', None, None, 6001]
19 ['PRINT', None, None, 15001]
20 ['PRINT', None, None, 14002]
21 ['PRINT', None, None, 12005]
22 ['PRINT', None, None, 12004]
23 [14, None, None, None]
```

## Pilas

### PilaO

Contiene los nombres de las variables pendientes de resolver dentro de operaciones aritméticas/comparación.

## PType

Igual que PilaO, pero contiene el tipo de la variable.

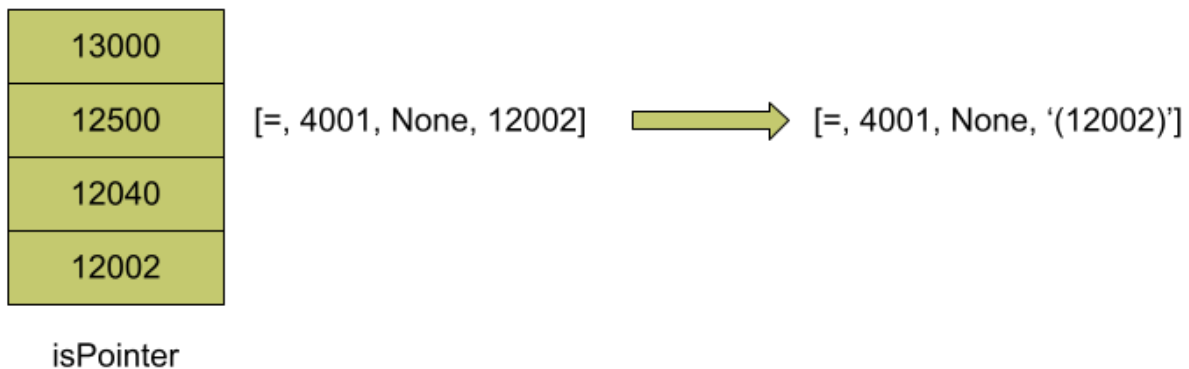
## POper

Contiene los símbolos de las operaciones aritméticas, de comparación, asignación, etc pendientes a resolver.

## IsPointer

Esta pila contiene las direcciones de memoria que apuntan hacia otra dirección. La utilizamos al procesar los cuádruplos para checar si alguna dirección pertenece a isPointer, en caso de ser verdad agregamos paréntesis a la dirección. Esta solución se implementó por qué, cómo mencioné en la sección de [PilaO](#), las operaciones pendientes se resuelven con los nombres de las variables, y para buscar la dirección de una variable por su nombre es necesario que esté en formato numérico, ya que el resto de las direcciones son de tipo INT y podríamos tener un error de compatibilidad.

Podemos observar un ejemplo en los diagramas de sintaxis de [operaciones aritméticas](#) y en el siguiente diagrama.



## AssignToArray

Cuando implementé las operaciones con apuntadores de arreglos, me dí cuenta de que la sintaxis resuelve el apuntador (VER s1, encontrar dirección, etc.) antes que la expresión a asignar. Ej **a[0] = 3\*2**: Si resolvemos y guardamos la dirección del arreglo en (12000), y al resolver la expresión guardamos el resultado en t0. Tendríamos las siguientes pilas:

**PilaO: [12000, t0]** y **POper: [=]** que resultaría en el siguiente cuádruplo: **[=, (12000), None t0]**

Este cuádruplo indica que estamos asignando la dirección a la variable t0.

Utilicé la pila de AssignToArray, para borrar el apuntador temporalmente (hasta que se resuelva la expresión) cuando se le está asignando un valor. Cómo podemos ver en el diagrama de [ASIGNACIONARR](#), empujamos el apuntador después del signo '='.

Ejemplo:

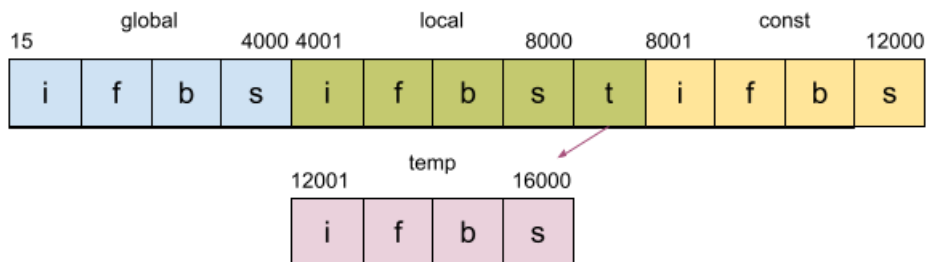
**PilaO: [12000] -> AssignToArray: [12000] POper: [=] -> PilaO: [t0, 12000] POper: [=]**

# Descripción de la Máquina Virtual

## Administración de Memoria

### Direcciones de Memoria

Las [direcciones](#) especificadas en la sección de Semántica, fueron utilizadas para crear el siguiente concepto de almacenamiento:



Este diagrama se traduce a tres objetos, Global, Local y Constantes con cuatro arreglos, Enteros, Flotantes, Booleanos y Strings. Solamente dentro de Local existe otro objeto llamado Temp, que contiene también los cuatro arreglos. Antes de realizar la ejecución, estos arreglos se inicializan con una lista de *None* del tamaño especificado en *resource\_count* dentro de la [tabla de funciones](#). Utilicé objetos para guardar las listas debido a la facilidad de legibilidad y por el costo de  $O(1)$  por acceder a los elementos del objeto.

Al momento de ejecución se realiza una búsqueda de cada dirección en el cuadruplo en el objeto de [direcciones](#) con la siguiente función:

```
def encontrarScope(self, dirVar):
    if dirVar in self.dp:
        return self.dp[dirVar]
    for x in self.vars:
        isTemp = False
        if x == 'temp':
            isTemp = True
        for i in self.vars[x]:
            dirs = self.vars[x][i]
            if(dirs[1]<=dirVar and dirVar<=dirs[2]):
                self.dp[dirVar] = (x, i, dirs[1], isTemp)
                # Returns scope, type, initial_dir, bool (if is a temp var)
                return x, i, dirs[1], isTemp
```

Con esta función podemos obtener e insertar valores en el objeto de almacenamiento descrito a continuación.

## Almacenamiento

global

int	[None] * (global[resource_count][int]+1)
float	[None] * (global[resource_count][float]+1)
bool	[None] * (global[resource_count][bool]+1)
string	[None] * (global[resource_count][string]+1)

local

int	[None] * (funcName[resource_count][int]+1)
float	[None] * (funcName[resource_count][float]+1)
bool	[None] * (funcName[resource_count][bool]+1)
string	[None] * (funcName[resource_count][string]+1)
temp	

temp

int	[None] * (funcName[resource_count][temp][int]+1)
float	[None] * (funcName[resource_count][temp][float]+1)
bool	[None] * (funcName[resource_count][temp][bool]+1)
string	[None] * (funcName[resource_count][temp][string]+1)

const

int	[None] * (const[resource_count][int]+1)
float	[None] * (const[resource_count][float]+1)
bool	[None] * (const[resource_count][bool]+1)
string	[None] * (const[resource_count][string]+1)

Decidí utilizar arreglos dinámicos para cada tipo de dato y cada ambiente (global, local, temp, const) con el propósito de que la dirección contenga esta información sobre el dato almacenado. También ahorramos espacio al almacenar solamente la cantidad de recursos utilizados en lugar de la cantidad total de espacios asignados para cada tipo de dato (985-999 espacios).

# Pruebas

## Arithmetic

```
test_aritmetic.pyst
1  start() {
2      var c: int;
3
4      function main(): void {
5          var l: int;
6          var w: int;
7          l=3;
8          w=8;
9          c=l*w;
10         print(c+50*3/(2*2)-6*w);
11     };
12 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
pamelalozano@YY2VPQ6GXC compiladores % python3 run.py aritmetic

-----PYSTACHIO-----

13.5

-----END-----
```



## While Loops

```
test_while.pyst
1  start() {
2      var a: int;
3      var c: int;
4
5      function main(): void {
6          a = 1;
7          c = 8;
8          while(c>a){
9              if(c+a<11) {
10                 print("yes");
11             } else {
12                 print("no");
13             };
14             a = a+1;
15         };
16     };
17 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
pamelalozano@YY2VPQ6GXC compiladores % python3 run.py while
-----PYSTACHIO-----
yes
yes
no
no
no
no
no
```

## Fibonacci Cíclico

```
test_fibonacci_ciclico.pyst
1  start() {
2      function fibonacci(var n: int; var size: int;): int {
3          var a: int;
4          var b: int;
5          var i: int;
6          var c: int;
7          a = 0;
8          b = 1;
9          if(n < 0) {
10             print("Wrong Input");
11             break 0-1;
12         };
13         if(n !== 1) {
14             break b;
15         };
16         for(i=1;i<n;i=i+1) {
17             c = a + b;
18             a = b;
19             b = c;
20         };
21         return b;
22     };
23
24     function main(): void {
25         print(fibonacci(12));
26     };
27 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

● pamelalozano@YY2VPQ6GXC compiladores % python3 run.py fibonacci\_ciclico

-----PYSTACHIO-----

144

-----END-----

## Fibonacci Recursivo

```
test_fibonacci_recursivo.pyst
1  start() {
2      function fibonacci(var n: int): int {
3          var res: int;
4          if(n < 0) {
5              print("Wrong Input");
6              break 0-1;
7          };
8          if(n != 0) {
9              break 0;
10         };
11         if((n != 1) !! (n != 2)) {
12             break 1;
13         };
14         res = fibonacci(n-1) + fibonacci(n-2);
15         return res;
16     };
17
18     function main(): void {
19         var n: int;
20         input(n);
21         print(fibonacci(n));
22     };
23 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

● pamelalozano@YY2VPQ6GXC compiladores % python3 run.py fibonacci\_recursivo  
Pystachio > 12

-----PYSTACHIO-----

144

-----END-----

## Sort (For anidado)

```
test_sort.pyst
1  start() {
2      var arr[5]: int;
3      var size: int;
4      function bubbleSort(var first: int; var second: int;): void {
5          var temp: int;
6          var i: int;
7          var j:int;
8          for(i=0-1;i<size-2;i=i+1) {
9              for(j=0-1;j<size-i-2;j=j+1) {
10                 if ((arr[j]) > (arr[j+1])){
11                     temp = arr[j];
12                     arr[j]=arr[j+1];
13                     arr[j+1]=temp;
14                 };
15             };
16         };
17         for(i=0-1;i<size-1;i=i+1) {
18             print(arr[i]);
19         };
20     };
21     function main(): void {
22         size = 5;
23         arr[0]=3;
24         arr[1]=5;
25         arr[2]=10;
26         arr[3]=60;
27         arr[4]=1;
28         bubbleSort();
29     };
30 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

● pamelalozano@YY2VPQ6GXC compiladores % python3 run.py sort

```
-----PYSTACHIO-----
1
3
5
10
60
-----END-----
```

## Find in Matrix

```

test_find_matrix.pyst
1  start() {
2      var arr[3][4]: int;
3      var size1: int;
4      var size2: int;
5      function find(var n: int): int {
6          var i: int;
7          var j: int;
8          for(i=0-1;i<size1-1;i=i+1) {
9              for(j=0-1;j<size2-1;j=j+1) {
10                 if ((arr[i][j]) != n){
11                     print(j, i);
12                     break 0;
13                 };
14             };
15         };
16         return (0-1);
17     };
18     function main(): void {
19         size1 = 3;
20         size2 = 4;
21         arr[0][0]=3;
22         arr[0][1]=70;
23         arr[0][2]=91;
24         arr[0][3]=43;
25         arr[1][0]=87;
26         arr[1][1]=2;
27         arr[1][2]=11;
28         arr[1][3]=49;
29         arr[2][0]=333;
30         arr[2][1]=10001;
31         arr[2][2]=203;
32         arr[2][3]=45;
33         find(11);
34     };
35 }

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```

● pamelalozano@YY2VPQ6GXC compiladores % python3 run.py find_matrix

-----PYSTACHIO-----

1
2

-----END-----

```

# Manual de Usuario

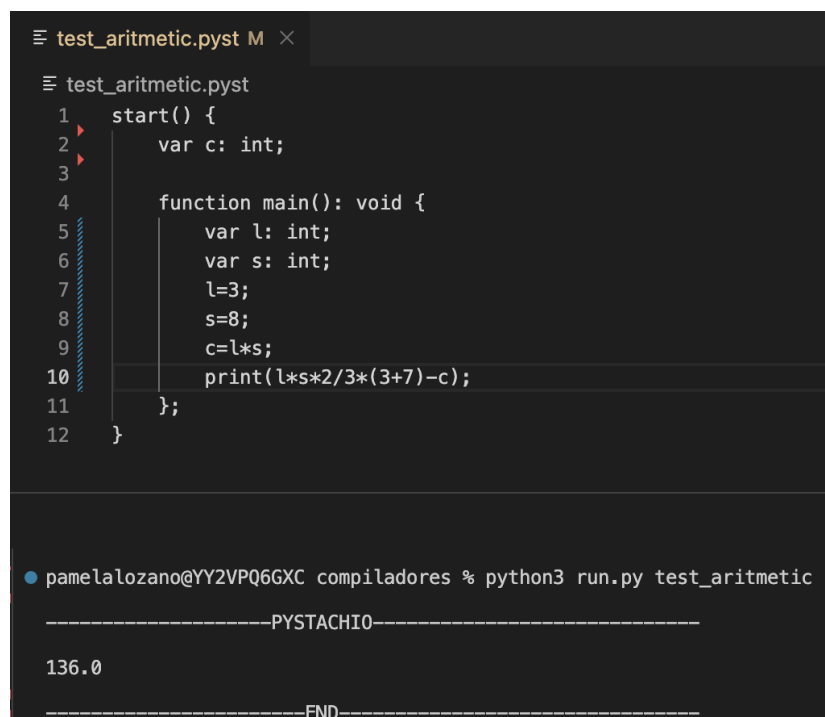
1. Instalar Python
  - a. [Guía de como instalar python](#)
2. Descargar zip de repositorio
  - a. Repository link: <https://github.com/pamelalozano16/compiladores>
3. Abrir zip y localizar ubicación del folder
4. Abrir terminal o CMD y mover ubicación al path del folder 'compiladores'.
5. Crear un archivo .pyst en la carpeta 'compiladores'.
6. Correr run.py en la terminal

```
python3 run.py {nombre de archivo sin extension}
```

## Video Demostración

[https://drive.google.com/file/d/1nd482O\\_-AZ4FoeYNxJCO6uZSS\\_SLKrqV/view?usp=share\\_link](https://drive.google.com/file/d/1nd482O_-AZ4FoeYNxJCO6uZSS_SLKrqV/view?usp=share_link)

## Ejemplo



```
test_aritmetic.pyst M x
test_aritmetic.pyst
1 start() {
2   var c: int;
3
4   function main(): void {
5     var l: int;
6     var s: int;
7     l=3;
8     s=8;
9     c=l*s;
10    print(l*s*2/3*(3+7)-c);
11  };
12 }

pamelalozano@YY2VPQ6GXC compiladores % python3 run.py test_aritmetic
-----PYSTACHIO-----
136.0
-----END-----
```