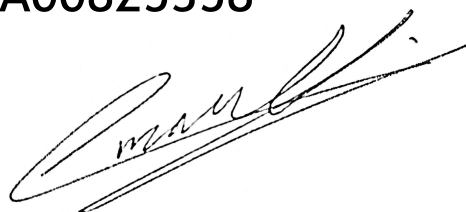


Objective-Lybad

24 de noviembre de 2021

Omar Ulises Montiel Escalante
A00825358



Índice

Índice	2
Descripción del Proyecto	4
Propósito y alcance del proyecto	4
Análisis de requerimientos y descripción de test cases	4
Lista de requerimientos	4
Test Case 1: Factorial cíclico	5
Test Case 2: Factorial recursivo	5
Test Case 3: Fibonacci cíclico	6
Test Case 4: Fibonacci recursivo	7
Test Case 5: Sort	8
Test Case 6: Find	9
Test Case 7: Operaciones sobre arreglos	10
Descripción de proceso de desarrollo	11
Bitácora General	11
Lista de commits	12
Reflexión	15
Descripción del Lenguaje	16
Nombre del lenguaje	16
Descripción de las características	16
Listado de errores	16
Compilación	16
Ejecución	18
Descripción del Compilador	18
Equipo de cómputo, lenguaje y utilerías	18
Análisis de léxico	18
Patrones de construcción	18
Enumeración de tokens	19
Análisis de sintaxis	21
Generación de código intermedio y análisis semántico	22

Código de operación	22
Diagramas de sintaxis	23
Acciones semánticas	31
Tabla de consideraciones semánticas	34
Administración de Memoria	34
Directorio de Funcionamiento	34
Cuádruplos	35
Pilas	36
Memoria Virtual	38
Descripción de la Máquina Virtual	39
Equipo de cómputo, lenguaje y utilerías	39
Administración de memoria	39
Pruebas del Funcionamiento del Lenguaje	40
Test Case 1: Factorial cíclico	40
Test Case 2: Factorial recursivo	41
Test Case 3: Fibonacci cíclico	43
Test Case 4: Fibonacci recursivo	44
Test Case 5: Sort	46
Test Case 6: Find	49
Test Case 7: Operaciones sobre arreglos	51

Descripción del Proyecto

Propósito y alcance del proyecto

El propósito de este proyecto es diseñar e implementar un lenguaje de programación con el fin de aplicar los conocimientos vistos durante el transcurso de la materia de Diseño de Compiladores TC3048.2.

Dentro del alcance se encuentra un compilador integrado por un analizador léxico y un analizador sintáctico, y a su vez una máquina virtual que ejecute el código intermedio generado por el compilador.

Análisis de requerimientos y descripción de test cases

Lista de requerimientos

- Análisis de léxico
- Análisis de sintaxis
- Análisis de semántica
 - Construir directorio de procedimientos
 - Construir tablas de variables
 - Construir tabla de consideraciones semánticas
- Generación de código
 - Generar código para expresiones aritméticas
 - Generar código para estatutos secuenciales
 - Generar código para estatutos condicionales
 - Generar código para funciones
 - Generar código para arreglos
- Máquina virtual
 - Ejecutar expresiones aritméticas
 - Ejecutar estatutos secuenciales
 - Ejecutar estatutos condicionales

- Ejecutar funciones
- Ejecutar arreglos

Test Case 1: Factorial cíclico

Este programa contiene una función que tiene un entero como parámetro y regresa el factorial de este, calculándolo de manera cíclica.

```

Program patito;
VARs
    int: a;

function int factorial(int n)
VARs
    int: i, result; {
    if (n <= 0) then {
        return(-1);
    }
    else {
        result = 1;
        for i = 1 to n + 1 do {
            result = result * i;
        }
        return(result);
    }
}

main() {
    write(factorial(4));
}

```

Test Case 2: Factorial recursivo

Este programa contiene una función que tiene un entero como parámetro y regresa el factorial de este, calculándolo de manera recursiva.

```

Program patito;
VARs
    int: a;

```

```

function int factorial(int n)
VARs
    int: i, result; {
    if (n < 0) then {
        return(-1);
    }
    else {
        if (n == 0) then {
            return(1);
        }
        else {
            return(n * factorial(n - 1));
        }
    }
}

main() {
    write(factorial(4));
}

```

Test Case 3: Fibonacci cíclico

Este programa contiene una función que tiene un entero como parámetro y regresa el número en la posición de este entero en la serie de Fibonacci, calculándolo de manera cíclica.

```

Program patito;
VARs
    int: a;

function int fibonacci(int n)
VARs
    int: i, n1, n2, result; {
    n1 = 1;
    n2 = 1;
    if (n <= 0) then {
        return(-1);
    }
    else {

```

```

    if (n == 1) then {
        return(n1);
    }
    else {
        if (n == 2) then {
            return(n2);
        }
        else {
            for i = 3 to n + 1 do {
                result = n1 + n2;
                n1 = n2;
                n2 = result;
            }
            return(result);
        }
    }
}

main() {
    write(fibonacci(8));
}

```

Test Case 4: Fibonacci recursivo

Este programa contiene una función que tiene un entero como parámetro y regresa el número en la posición de este entero en la serie de Fibonacci, calculándolo de manera recursiva.

```

Program patito;
VARs
    int: a;

function int fibonacci(int n) {
    if (n <= 0) then {
        return(-1);
    }
    else {
        if (n == 1) then {

```

```

        return(1);
    }
    else {
        if (n == 2) then {
            return(1);
        }
        else {
            return(fibonacci(n - 1) + fibonacci(n - 2));
        }
    }
}

main() {
    write(fibonacci(8));
}

```

Test Case 5: Sort

Este programa contiene una función que ordena un arreglo unidimensional de manera ascendente.

```

Program MyRlike;
VARS
    int: arreglo[10], n, i, j;

function void sort()
VARS
    int: temp; {
    for i = 0 to n - 1 do {
        for j = 0 to n - i - 1 do {
            if (arreglo[j] > arreglo[j + 1]) then {
                temp = arreglo[j];
                arreglo[j] = arreglo[j + 1];
                arreglo[j + 1] = temp;
            }
        }
    }
}

```



```

main() {
    n = 10;
    arreglo[0] = 52;
    arreglo[1] = 36;
    arreglo[2] = 85;
    arreglo[3] = 12;
    arreglo[4] = 48;
    arreglo[5] = 96;
    arreglo[6] = 35;
    arreglo[7] = 31;
    arreglo[8] = 28;
    arreglo[9] = 95;

    sort();

    for i = 0 to n do {
        write(arreglo[i]);
    }
}

```

Test Case 6: Find

Este programa contiene una función que recibe como parámetro un entero y regresa el índice de la posición donde este entero se encuentra dentro de un arreglo.

```

Program MyRlike;
VARs
    int: arreglo[10], n, i;

function int find(int x)
VARs
    int: index; {
    index = -1;
    for i = 0 to n do {
        if (arreglo[i] == x) then {
            index = i;
        }
    }
}

```

```

    return(index);
}

main() {
    n = 10;
    arreglo[0] = 52;
    arreglo[1] = 36;
    arreglo[2] = 85;
    arreglo[3] = 12;
    arreglo[4] = 48;
    arreglo[5] = 96;
    arreglo[6] = 35;
    arreglo[7] = 31;
    arreglo[8] = 28;
    arreglo[9] = 95;

    write(find(35));
}

```

Test Case 7: Operaciones sobre arreglos

Este programa contiene una función que accede los contenidos de dos arreglos y los multiplica guardando el resultado en un tercer arreglo.

```

Program MyRlike;
VARs
    int: arreglo1[5], arreglo2[5], arreglo3[5], n, i;

function void multiply() {
    for i = 0 to n do {
        arreglo3[i] = arreglo1[i] * arreglo2[i];
    }
}

main() {
    n = 5;

    arreglo1[0] = 7;
    arreglo1[1] = 3;

```

```

arreglo1[2] = 5;
arreglo1[3] = 9;
arreglo1[4] = 1;

arreglo2[0] = 4;
arreglo2[1] = 8;
arreglo2[2] = 2;
arreglo2[3] = 6;
arreglo2[4] = 10;

multiply();

for i = 0 to n do {
    write(arreglo3[i]);
}
}

```

Descripción de proceso de desarrollo

Bitácora General

- Avance 1 (1 de octubre)
 - Se construyeron los diagramas necesarios para la sintaxis del lenguaje
 - Se diseñó la gramática y el RegEx necesario
 - Se desarrolló el código para el analizador léxico y sintáctico
 - Se implementaron tres pruebas
- Avance 2 (11 de octubre)
 - Se reestructuró la gramática para facilitar la construcción del Directorio de Procedimiento
 - Se agregó el Directorio de Procedimiento y las Tablas de las Variables
- Avance 3 (15 de octubre)
 - Se creó la tabla de consideraciones semánticas (cubo semántico)
- Avance 4 (12 de octubre)
 - Se modificó la gramática para permitir escribir expresiones como estatutos (aunque se pierda la información)

- Se empezó a almacenar expresiones, construyendo las bases para luego generar el código de expresiones aritméticas
- Avance Final (24 de noviembre)
 - Se implementó generación de código de expresiones aritméticas
 - Se implementó generación de código de estatutos secuenciales
 - Se implementó generación de código de estatutos condicionales
 - Se implementó generación de código de funciones
 - Se implementó generación de código de arreglos
 - Se añadieron tests
 - Se creó máquina virtual

Lista de commits

- commit d4a2c69bb4016f5e9fd12c404816df0a02d87a27 (HEAD -> master, origin/master)
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 23:17:22 2021 -0600
 - Add documentation comments for virtual machine
- commit 33ba3bc71bc66636bd301153d0136ab71630ed4a
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 22:28:31 2021 -0600
 - Add virtual machine functionality for arithmetic expressions
- commit 5efcf9bc2766bebc308222fe1ccc1cc557d7fb9
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 21:27:59 2021 -0600
 - Create memory instantiation functions and store constants from code generation
- commit 841a71c8ade9958f923d7c3f23571cdb082db31d
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 20:28:04 2021 -0600
 - Add documentation comments
- commit 189d151b2eed057c7ed60be97209da183b74c301
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 20:01:57 2021 -0600
 - Fix error with write statement

- commit 6ba3a3ebccc4d7b09930df7f0f36d6c03c12c0c1
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 13:16:20 2021 -0600
 - Fix issues with virtual memory not being erased after functions end
- commit bd4a17e0aa3f46e81b11b905b0eeace84f502de7
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 12:42:38 2021 -0600
 - Serialize data to json and create virtual machine
- commit 4c606ff30f950d09c2d8b23398b52128e458a51b
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 12:06:51 2021 -0600
 - Add required tests
- commit a218e8812f7377d25ecb6d8f8a54d8cb5dddcc48
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 24 01:03:46 2021 -0600
 - Add quadruple generation for arrays and add virtual memory
- commit 2b2f27a4d53b13c626437ce876d829385e01898e
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Tue Nov 23 13:29:34 2021 -0600
 - Add quadruple generation for functions
- commit 55e5b97c33fca0a9d2f932090817fbc3eb4c3287
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Wed Nov 17 18:53:11 2021 -0600
 - Add quadruple generation for conditional statements
- commit 9a7dda28be90b218ec6e964010cb45fb3bfd6b92
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Tue Nov 16 12:21:52 2021 -0600
 - Add quadruple generation for arithmetic expressions and sequential statements
- commit 68b3d16b1a0f0828a5e8e2587d67eb0803157c84
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 22 16:41:14 2021 -0700
 - Start to store statements to eventually build code for expressions
- commit fcb3aa6df186b23453eb188af9c148f4ace281e3

- Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 15 15:30:40 2021 -0700
 - Add Semantic Cube
- commit 67d44c0191e32f28b46a2a018a2bd9c7681da5c7
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Mon Oct 11 10:45:09 2021 -0700
 - Update README.md
- commit 42d46623795c4d9844a180b4eed50e0495d8f19b
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Sun Oct 10 17:30:45 2021 -0700
 - Fix bug with adding void functions to Function Directory
- commit 2048f378e99b935066b8721d7d73414fe993f4e7
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Sun Oct 10 17:16:24 2021 -0700
 - Add Function Directory and Variable Tables
- commit f09fb8428abaa43211aa34ac7e81b332b2620580
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 8 13:19:15 2021 -0700
 - Add Syntax-Directed Translation
- commit fc7238f130ae5b25be44c2548b69cf9490fbc93
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 20:50:45 2021 -0700
 - Add first delivery progress details
- commit 906d7ae9556d51fde2ddf8aa4ecca53f19f34245
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:55:02 2021 -0700
 - Add tests
- commit 68976741065d830b5e01f80f8ac0dd60eb5267d3
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:51:33 2021 -0700
 - Add lexer and parser
- commit b76a5d989fa1c487133f77694f83e00701695b0b
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:41:54 2021 -0700

- Fix regex in markdown
- commit 0019714e0a6c1676eb6f9808a22b1ae4cc60d390
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:32:19 2021 -0700
 - Fix grammar and regex markdown
- commit e695b52625b69c3d05d1b742587e9fd507febbc8
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:26:49 2021 -0700
 - Add grammar and regex
- commit 347e6f80846a899840bfc8860557f670a749d395
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:17:49 2021 -0700
 - Add diagrams
- commit a4904956a6392f53e127ed32e6512770a0b38197
 - Author: OUMontiel <omarume@gmail.com>
 - Date: Fri Oct 1 14:17:19 2021 -0700
 - Add README

Reflexión

Puedo decir que esta es la materia más retadora que he tenido en el transcurso de la carrera, debido a la cantidad de partes que componen e interactúan en la creación de un lenguaje de programación, desde empezar creando el léxico y la sintaxis en diagramas de flujo, hasta ejecutar el código intermedio generado por el compilador en una máquina virtual. Aprendí cómo es que el compilador primero analiza el léxico, seguido por la sintaxis, y en tercer lugar a la semántica. También apliqué el método de cuádruplos para la generación de código intermedio visto en el curso, y pude asimilar de mejor manera cómo es que la memoria de la computadora funciona al simular una memoria virtual.

Desafortunadamente, tuve un semestre personalmente difícil, por lo que se me complicó seguir la lista sugerida de avances, y tuve que realizar la mayoría del proyecto al final del curso. Me hubiera gustado dedicarle más tiempo al proyecto para hacerlo de manera más estructurada y limpia, y a la vez mejor documentado con comentarios. Sin embargo, considero que lo que realicé sí me brindó los conocimientos que la materia ofrece y me resultaron en su mayor parte muy interesantes y entretenidas.



Descripción del Lenguaje

Nombre del lenguaje

Objective-Lybad

Descripción de las características

Objective-Lybad es un lenguaje de programación básico y parecido a C, en el sentido que no maneja variables booleanas, sino que todo lo maneja con enteros. Los tipos de datos que maneja son enteros, flotantes, y caracteres. También se pueden crear arreglos unidimensionales de cualquiera de estos tres tipos y strings (que son en esencia un arreglo de caracteres).

El lenguaje tiene soporte para estatutos secuenciales (asignaciones, llamadas, retornos, escrituras y lecturas) y condicionales (*if*, *else*, *while* y *for*). Además, se pueden agregar funciones antes del *main*, que reciben parámetros, declaran variables locales y tienen la posibilidad de regresar variables. El *main* es la función principal que se ejecuta al iniciar el programa.

Listado de errores

Compilación

- Error de léxico

```
'LEXER ERROR: ', t.value
```


- Error de sintaxis

```
'PARSER ERROR: ', p
```

- Error de tipo de archivo

```
'ERROR: Text file expected!'
```

- Error de archivo inexistente

```
'ERROR: File does not exist!'
```

- Error de acceso de arreglo

```
'ERROR: Array size has to be of type integer!'
```

- Error de arreglo no identificado

```
'ERROR: Variable < ', p[1], ' > is not an array!'
```

- Error por falta de memoria

```
'ERROR: Stack overflow!'
```

- Error por redeclaración de variables

```
'ERROR: Variable name < ', parameterIDsStack[-1], ' > already exists!'
```

- Error por redeclaración de funciones

```
'ERROR: Function name < ', functionIDs[-1], '> already exists!'
```

- Error por uso de variables no declaradas

```
'ERROR: Variable < ', variable, ' > not found!'
```

- Error por discordancia de tipos

```
'ERROR: Types mismatch!'
```

- Error por tipo incorrecto de retorno

```
'ERROR: Return type does not match function type!'
```

- Error por tipo en expresión de *for*

```
'ERROR: \'For\' expression is not of type integer!'
```

- Error por uso de funciones no declaradas

```
'ERROR: Function < ', functionName, ' > does not exist!'
```

- Error por mal uso de la firma de una función

```
'ERROR: Parameter type does not match function signature!'
```

- Error por no introducir archivo de texto

```
'ERROR: Text file expected!'
```

- Error por introducir archivo de texto que no existe

```
'ERROR: File does not exist!'
```

Ejecución

- Error por utilizar variables que no están asignadas en memoria

```
'ERROR: Variables have not been assigned!'
```

- Error por no introducir archivo JSON

```
'ERROR: JSON file expected!'
```

- Error por introducir archivo JSON que no existe

```
'ERROR: File does not exist!'
```

Descripción del Compilador

Equipo de cómputo, lenguaje y utilerías

Objective-Lybad se desarrolló en Python, implementando el analizador léxico y el analizador sintáctico con la librería de PLY. También utiliza la librería de JSON para serializar el directorio de funcionamiento y los cuádruplos de la generación de código.

Análisis de léxico

Patrones de construcción

Token	RegEx
id	<code>[a-zA-Z_][a-zA-Z_0-9]*</code>
int	<code>\d+</code>

float	<code>\d+\.\d+</code>
char	<code>\'([^\']* \'\\\' \'\\\\\')?\'</code>
string	<code>\"([^\"]*" \"\\\" \"\\\\\")*\"</code>

Enumeración de tokens

Token	Valor
PROGRAM	Program
MAIN	main
VARs	VARs
FUNCTION	function
RETURN	return
READ	read
WRITE	write
IF	if
THEN	then
ELSE	else
WHILE	while
DO	do
FOR	for
TO	to
TYPEINT	int
TYPEFLOAT	float
TYPECHAR	char

TYPEVOID	void
SEMICOLON	;
COLON	:
COMMA	,
OPENCURLY	{
CLOSECURLY	}
OPENBOX	[
CLOSEBOX]
OPENPAR	(
CLOSEPAR)
IS	=
OR	
AND	&&
EQ	==
NE	!=
LT	<
LE	<=
GT	>
GE	>=
PLUS	+
MINUS	-
MULTIPLY	*
DIVIDE	/

MODULO	%
--------	---

Análisis de sintaxis

```

PROGRAMA → Program PROGRAMA1 PROGRAMA2 main() { PROGRAMA3 }
PROGRAMA1 → id ; VARS | id ;
PROGRAMA2 → FUNCIONES | ε
PROGRAMA3 → ESTATUTOS | ε
VARS → 'VARS' VARS1
VARS1 → TIPO : VARS2
VARS2 → id VARS3
VARS3 → [ cte-1 ] VARS4 | VARS4
VARS4 → ; VARS5 | COMMA VARS2
VARS5 → VARS1 | ε
LISTA_IDS → id [ EXP ] LISTA_IDS1 | id LISTA_IDS2
LISTA_IDS1 → COMMA LISTA_IDS | ε
TIPO → int | float | char
FUNCIONES → function FUNCIONES1 | ε
FUNCIONES1 → TIPO FUNCIONES2 FUNCIONES3 | void FUNCIONES2 FUNCIONES3
FUNCIONES2 → id (
FUNCIONES3 → PARAMETERS FUNCIONES4 | FUNCIONES4
FUNCIONES4 → ) FUNCIONES5
FUNCIONES5 → VARS { FUNCIONES6 | { FUNCIONES6
FUNCIONES6 → ESTATUTOS FUNCIONES6 | } FUNCIONES
PARAMETROS → TIPO id PARAMETERS1
PARAMETROS1 → , PARAMETERS | ε
ESTATUTOS → ASIGNACION ; ESTATUTOS1 | LLAMADA ; ESTATUTOS1 | RETORNO ;
ESTATUTOS1 | LECTURA ; ESTATUTOS1 | ESCRITURA ; ESTATUTOS1 | CONDICION
ESTATUTOS1 | WHILE ESTATUTOS1 | FOR ESTATUTOS1 | EXPRESION ;
ESTATUTOS1
ESTATUTOS1 → ESTATUTOS | ε
ASIGNACION → ASIGNACION1 EXPRESION ;
ASIGNACION1 → id [ EXP ] = | id =
LLAMADA → FUNCTION ;
FUNCTION → id ( FUNCION1
FUNCION1 → EXP FUNCION2 | )
FUNCION2 → , FUNCION1 | )
RETORNO → return ( EXP )

```

```

LECTURA → read ( LISTA_IDS )
ESCRITURA → write ( ESCRITURA1 )
ESCRITURA1 → cte.string ESCRITURA2 | EXPRESION ESCRITURA2
ESCRITURA2 → , ESCRITURA1 | ε
CONDICION → if ( EXPRESION ) then { ESTATUTOS } CONDICION1
CONDICION1 → else { ESTATUTOS } | ε
WHILE → while ( EXPRESION ) do { ESTATUTOS }
FOR → for id FOR1 = EXP to EXP do { ESTATUTOS }
FOR1 → id [ EXP ] | id
EST_EXP → EXPRESION ;
EXPRESION → AND | AND EXPRESION1 EXPRESION
EXPRESION1 → ||
AND → EQUAL | EQUAL AND1 AND
AND1 → '&&'
EQUAL → COMPARE | COMPARE EQUAL1 EQUAL
EQUAL1 → == | !=
COMPARE → EXP | EXP COMPARE1 COMPARE
COMPARE1 → < | <= | > | >=
EXP → TERMINO | TERMINO EXP1 EXP
EXP1 → + | -
TERMINO → FACTOR | FACTOR TERMINO1 TERMINO
TERMINO1 → * | / | %
FACTOR → FACTOR1 | ( EXPRESION ) | FUNCION | FACTOR2 cte
FACTOR1 → id [ EXP ] | id
FACTOR2 → + | - | ε
VARCTE → id | cte-l | cte-f | cte-c

```

Generación de código intermedio y análisis semántico

Código de operación

[\[operador, operando_izquierdo, operando_derecho, temporal\]](#)

Realiza la operación <operando_izquierdo operador operando_derecho> y asigna el resultado en temporal

[\[=, operando_derecho, None, operando_izquierdo\]](#)

Realiza la asignación <operando_izquierdo = operando_derecho>

[\[return, None, None, operando_retorno\]](#)

Regresa <operando_retorno> al final de una función

[read, None, None, operando_lectura]

Lee de consola y asigna en <operando_lectura>

[write, tamaño, None, operando_escritura]

Escribe en consola <operando_escritura> usando tamaño para saber dónde acaban los caracteres del string

[write, None, None, operando_escritura]

Escribe en consola <operando_escritura>

[gotof, operando_condicion, None, contador]

Si el <operando_condicion> es falso, salta al cuádruplo en <contador>

[goto, None, None, contador]

Salta al cuádruplo en <contador>

[era, función, None, None]

Crea la memoria local para <función>

[param, param_valor, None, param_número]

Manda <param_valor> a la memoria de la función actual en <param_número>

[gosub, función, None, temporal]

Crea la memoria local para <función>

[ver, operando, límite_inferior, límite_superior]

Verifica que <operando> sea mayor o igual a <límite_inferior> y menor a <límite_superior>

[endfunc, None, None, None]

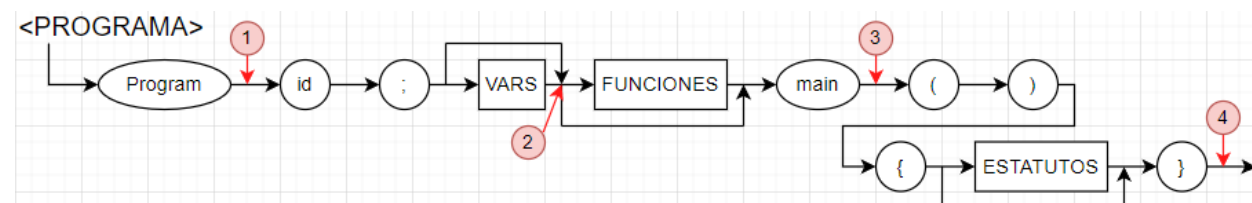
Termina la declaración de los estatutos de una función

[end, None, None, None]

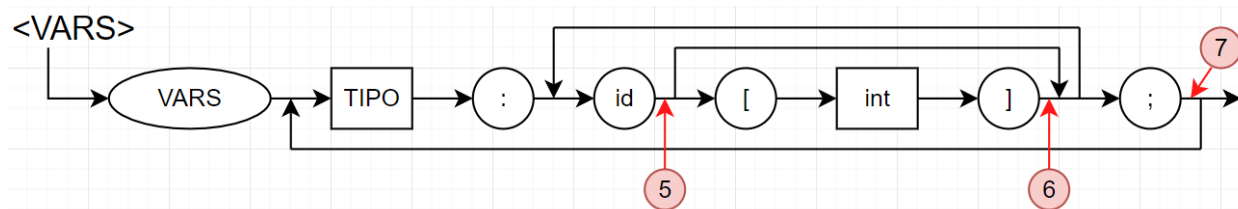
Termina el programa

Diagramas de sintaxis

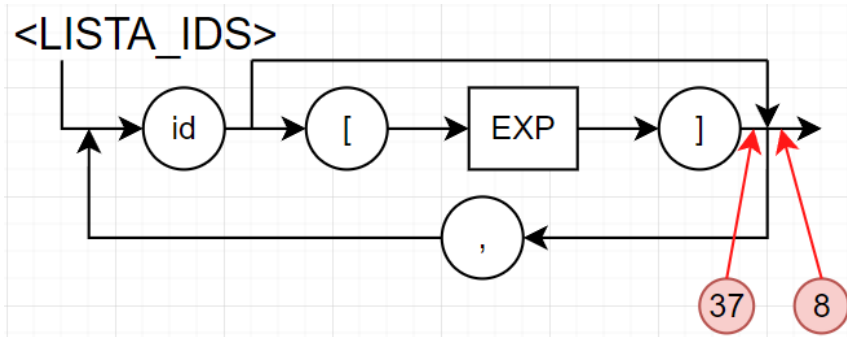
PROGRAMA



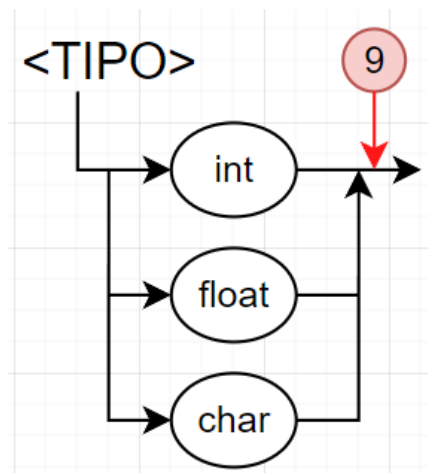
VARS



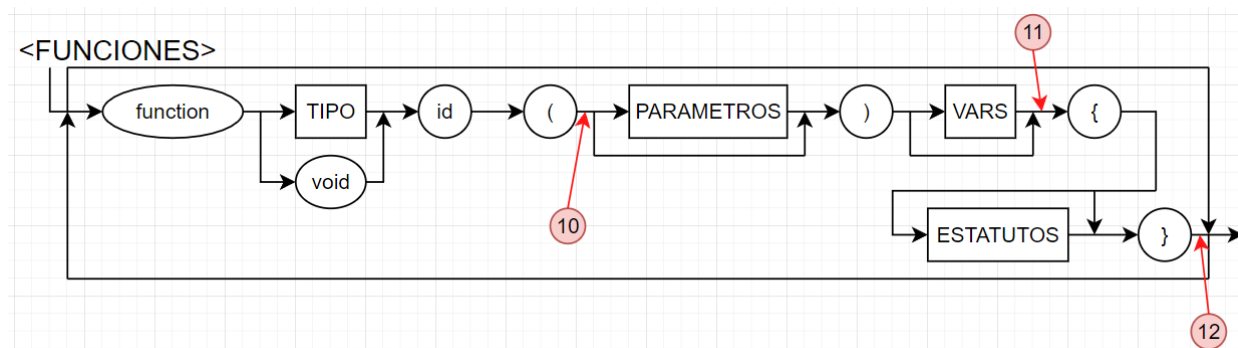
LISTA_IDS



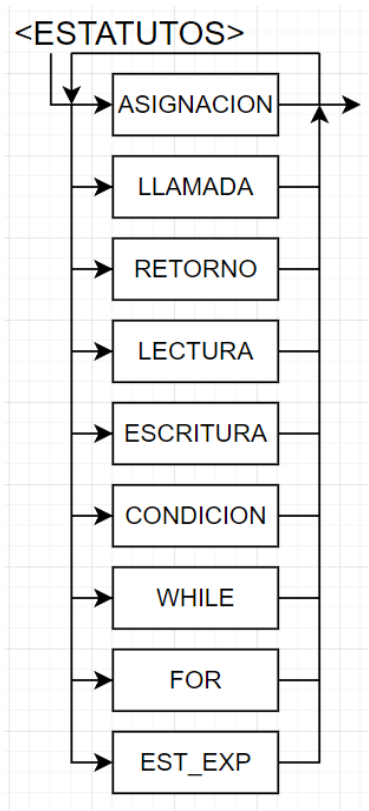
TIPO



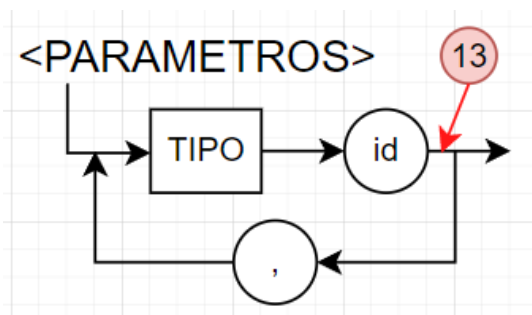
FUNCIONES



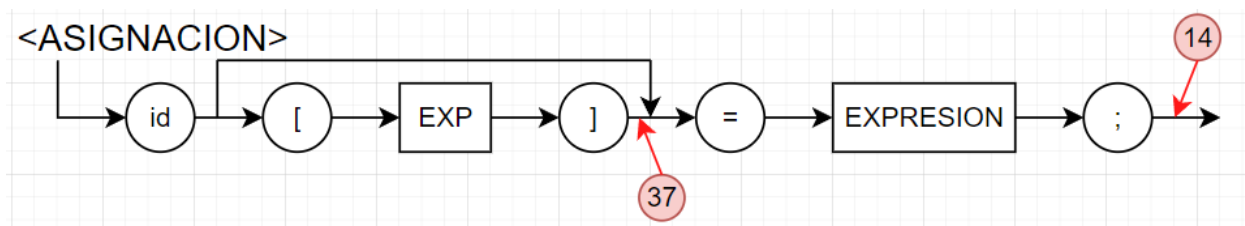
ESTATUTOS



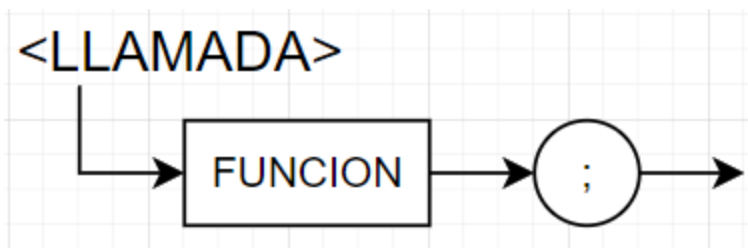
PARAMETROS



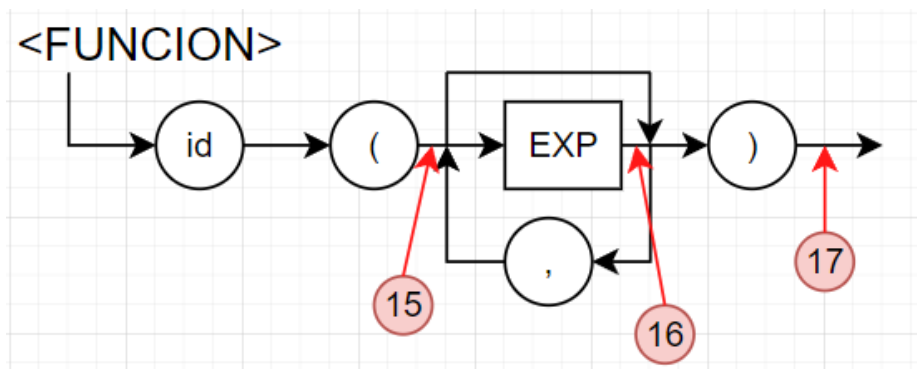
ASIGNACION



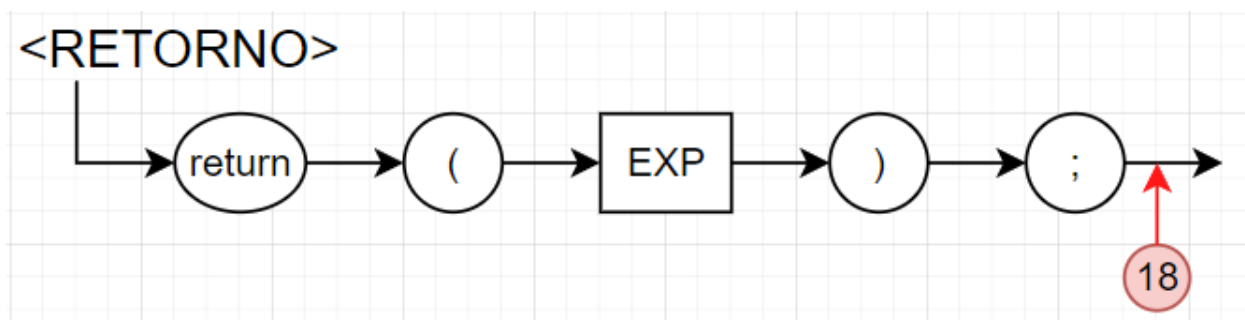
LLAMADA



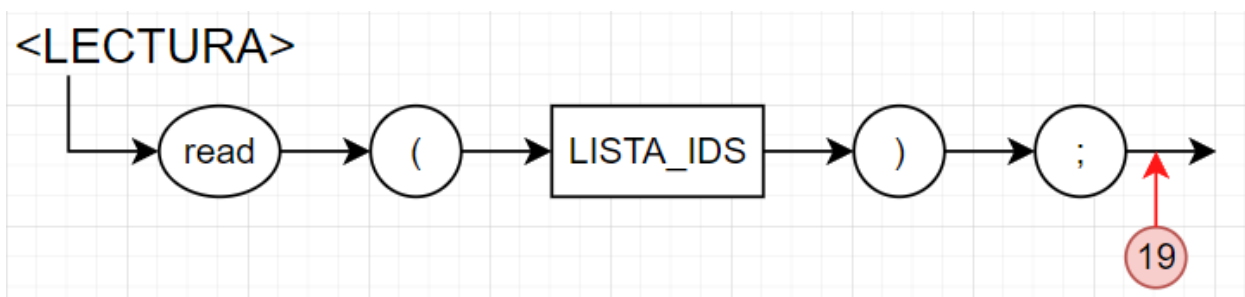
FUNCION



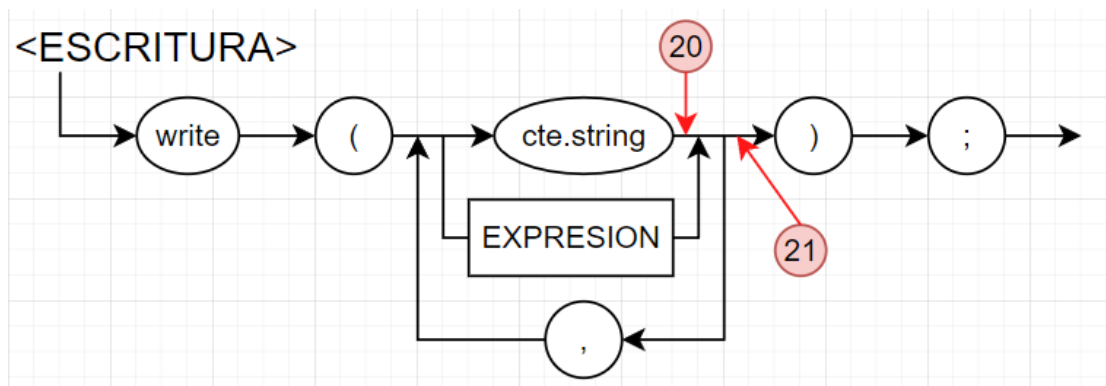
RETORNO



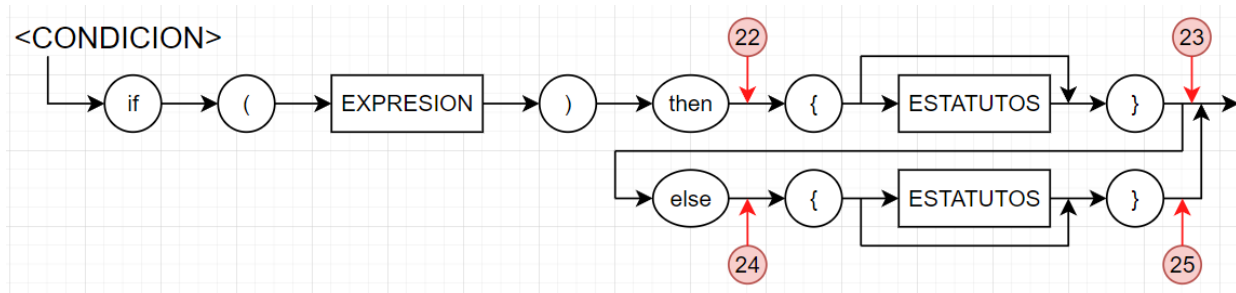
LECTURA



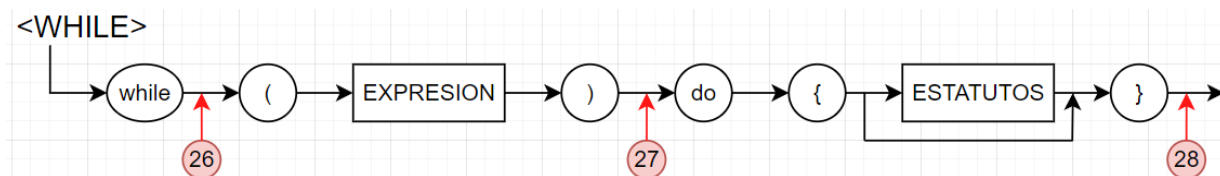
ESCRITURA



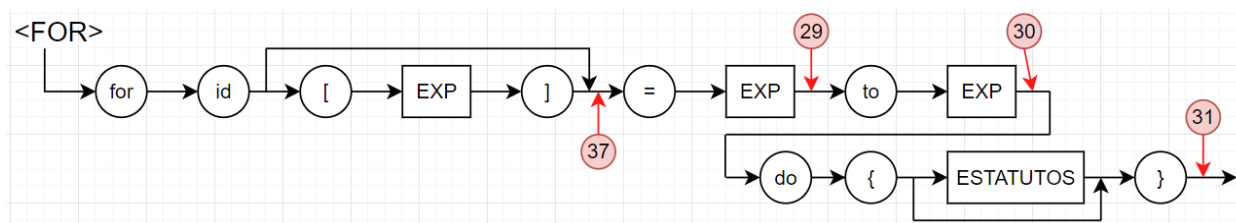
CONDICION



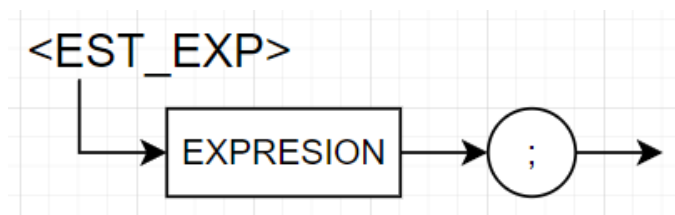
WHILE



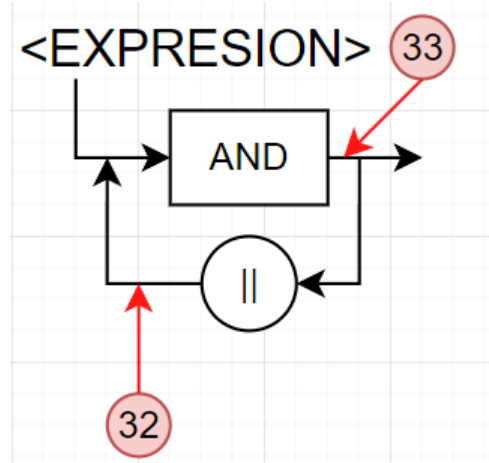
FOR



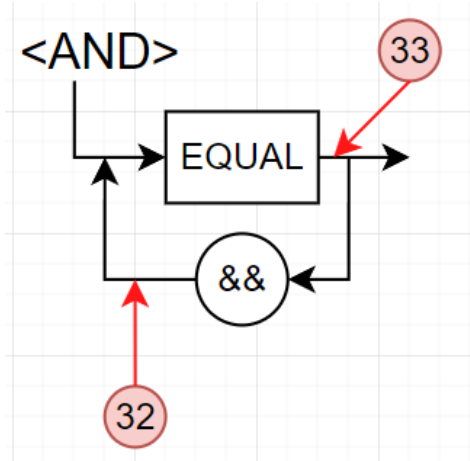
EST_EXP



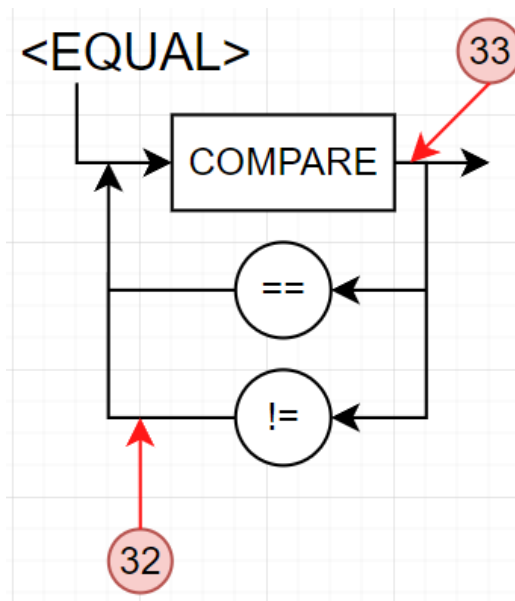
EXPRESION



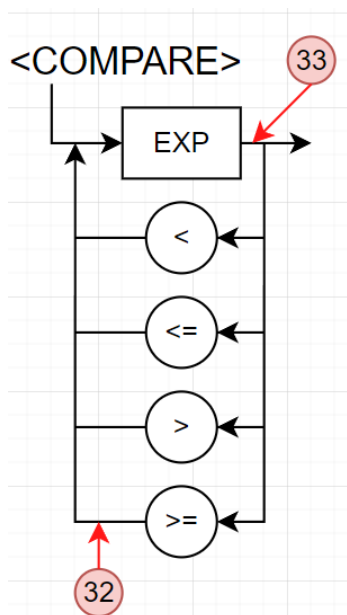
AND



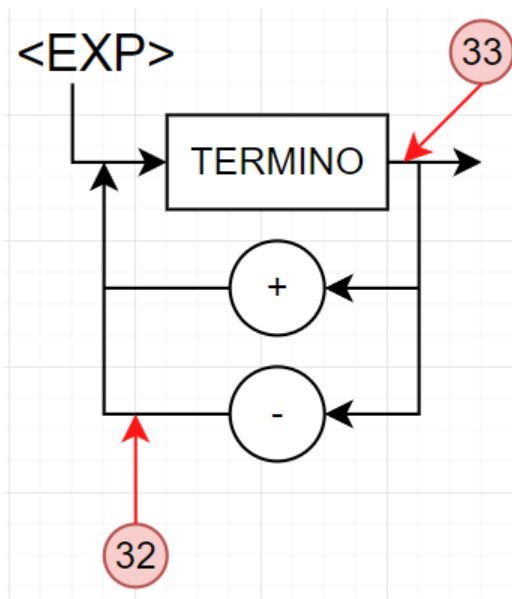
EQUAL



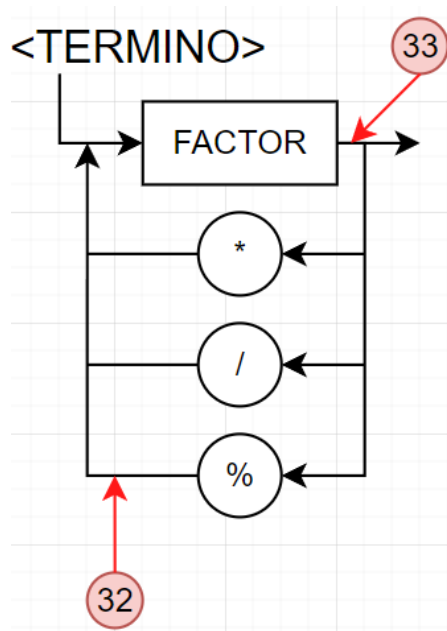
COMPARE



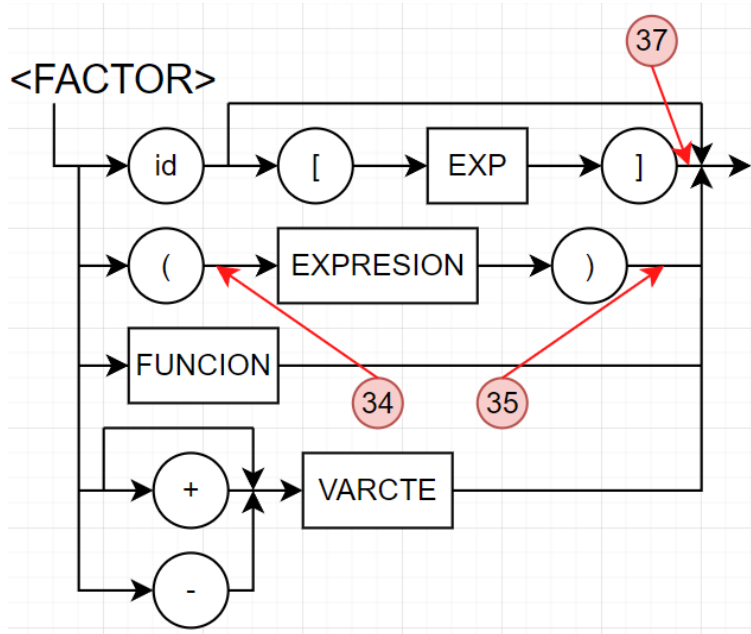
EXP



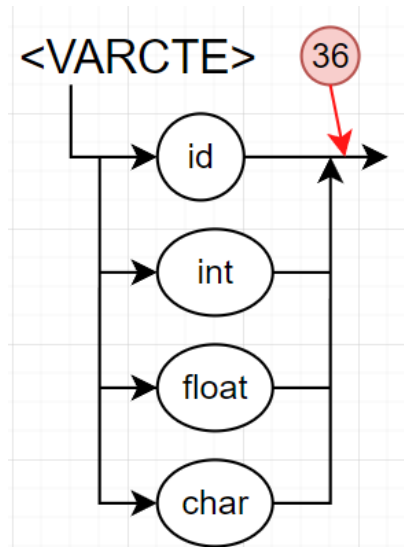
TERMINO



FACTOR



VARCTE



Acciones semánticas

1. Guardar [*goto*, *main*, *None*, *_*]
2. Construir tabla de variables
 - Añadir nombre de programa a *functionIDs*
 - Añadir tipo de función/programa a *functionKinds*
 - Añadir tabla de variables a *variablesTables*
 - Guardar función en directorio de funcionamiento
3. Llenar el campo de dirección del primer cuádruplo a cantidad de cuádruplos
4. Guardar [*end*, *None*, *None*, *None*]
 - Borrar memoria local y global
5. Añadir nombre de variable a *variableIDsStack*
6. Añadir tamaño de variable (o *None* en caso de no ser arreglo) a *variableSizesStack*
7. Añadir cantidad de variables a *variableTypesCountStack*
8. Añadir *read* a *operators*
9. Añadir tipo a *typesStack*
10. Añadir nombre de función a *functionIDs*
11. Construir tabla de variables
 - Añadir tipo de función/programa a *functionKinds*
 - Añadir tabla de parámetros a *parametersTables*
 - Añadir tabla de variables a *variablesTables*
 - Guardar función en directorio de funcionamiento

12. Guardar [*endfunc*, *None*, *None*, *None*]
Borrar memoria local
13. Añadir nombre de parámetro a *parameterIDsStack*
14. Sacar dos operators, dos types y un operators
Si tipos combinan, guardar [*operador*, *operador_derecho*, *None*,
operador_izquierdo]
15. Guardar [*era*, *ID*, *None*, *None*]
Añadir información de función a *functionNames*, *functionParametersTables* y
functionParametersIndex
16. Sacar un operators y un types
Si tipo no da error, guardar [*param*, *operador*, *None*, *param_índice*]
17. Guardar [*gosub*, *nombre_funcion*, *None*, *None*]
Si tipo de función no es void, guardar [*=*, *nombre_funcion*, *None*, *temp*]
Añadir *temp* a *operands* y *types*
18. Sacar un operands y un types
Si tipo no da error, guardar [*return*, *None*, *None*, *operando*]
19. Sacar operand, type y operator
Guardar [*operator*, *None*, *None*, *operand*]
20. Añadir string a *operands* y *types*
21. Sacar un operands y un types
Guardar [*write*, *None*, *None*, *operando*]
22. Sacar un operands y un types
Añadir contador a *jumps*
Guardar [*gotof*, *operando*, *None*, *_*]
23. Sacar un jumps
Llenar salto en *gotof*
24. Sacar un jumps
Añadir contador a *jumps*
Guardar [*goto*, *None*, *None*, *_*]
Llenar salto en *gotof*
25. Sacar un jumps
Llenar salto en *goto*
26. Añadir contador a *jumps*

27. Sacar un operands y un types

Guardar [*gotof*, *operando*, *None*, *_*]

28. Sacar dos jumps

Guardar [*goto*, *None*, *None*, *salto_retorno*]

Llenar salto en *gotof*

29. Sacar dos operands y dos types

Añadir *operando_control* a *controlVariables*

Si *tipo_exp* es int, guardar [*=*, *operando_exp*, *None*, *operando_control*]

30. Sacar un operands y un types

Añadir contador a jumps

Guardar [*<*, *variable_control*, *operando*, *temp*]

Añadir contador a jumps

Guardar [*gotof*, *temp*, *None*, *_*]

Añadir *temp* a operands y types

31. Guardar [*+*, *variable_control*, *1*, *variable_control*]

Sacar dos de jumps

Guardar [*goto*, *None*, *None*, *salto_retorno*]

Llenar contador en *gotof*

32. Añadir operador a operators

33. Sacar dos operands, dos types y un operators

Si no hay error de operación, guardar [*operador*, *operando_izquierdo*, *operando_derecho*, *temp*]

Añadir *temp* a operands y types

34. Añadir paréntesis a operators

35. Sacar un operators

36. Añadir constante/variable a operands y types

37. Si no es arreglo, añadir ID a operands y types

De lo contrario

Sacar un operands y un types

Guardar [*ver*, *operando*, *0*, *tamaño_arreglo*]

Guardar [*+*, *operando*, *direccion_arreglo*, *temp*]

Añadir *temp* a operands y types

Tabla de consideraciones semánticas

Esta es una estructura de tres dimensiones donde en las primeras dos dimensiones se especifican los tipos de datos de las variables que funcionan como operandos y la tercera dimensión especifica el operador de la expresión. El contenido de cada celda contiene el tipo de dato resultante de la expresión, o error de ser el caso.

Este “cubo” semántico construido se desglosa de la siguiente manera:

INT													
		&&	==	!=	<	<=	>	>=	+	-	*	/	%
INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT
FLOAT	INT	INT	INT	INT	INT	INT	INT	INT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT
CHAR	INT	INT	INT	INT	INT	INT	INT	INT	CHAR	CHAR	CHAR	CHAR	CHAR

FLOAT													
		&&	==	!=	<	<=	>	>=	+	-	*	/	%
INT	INT	INT	INT	INT	INT	INT	INT	INT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT
FLOAT	INT	INT	INT	INT	INT	INT	INT	INT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT
CHAR	INT	INT	INT	INT	INT	INT	INT	INT	error	error	error	error	error

CHAR													
		&&	==	!=	<	<=	>	>=	+	-	*	/	%
INT	INT	INT	INT	INT	INT	INT	INT	INT	CHAR	CHAR	CHAR	CHAR	CHAR
FLOAT	INT	INT	INT	INT	INT	INT	INT	INT	error	error	error	error	error
CHAR	INT	INT	INT	INT	INT	INT	INT	INT	CHAR	CHAR	CHAR	CHAR	CHAR

Administración de Memoria

Directorio de Funcionamiento

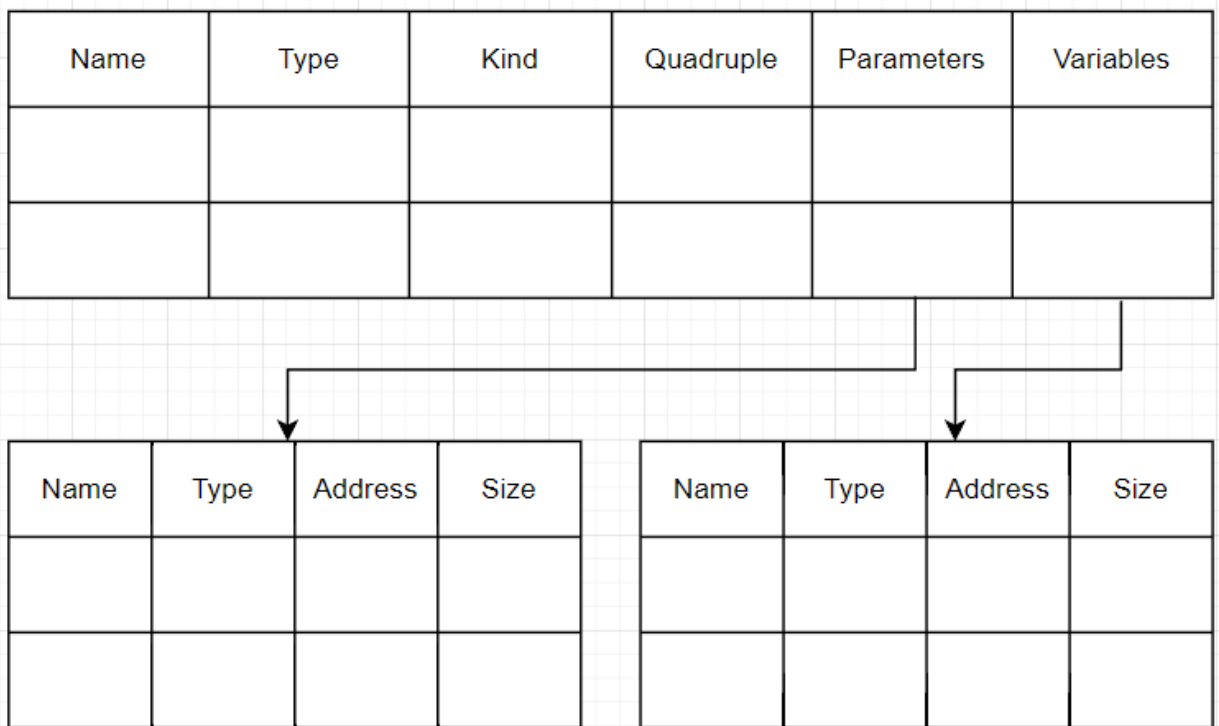
El Directorio de Funcionamiento es un diccionario que contiene seis llaves:

- Name: Nombre de la función
- Type: Tipo de retorno de la función
- Kind: Determina si es función o programa global

- Quadruple: El número de cuádruplo en el que se encuentra su primer estatuto
- Parameters: Tabla de variables que contiene los parámetros de la función (*None* si es programa global)
- Variables: Tabla de variables (locales si es función y globales si es programa)

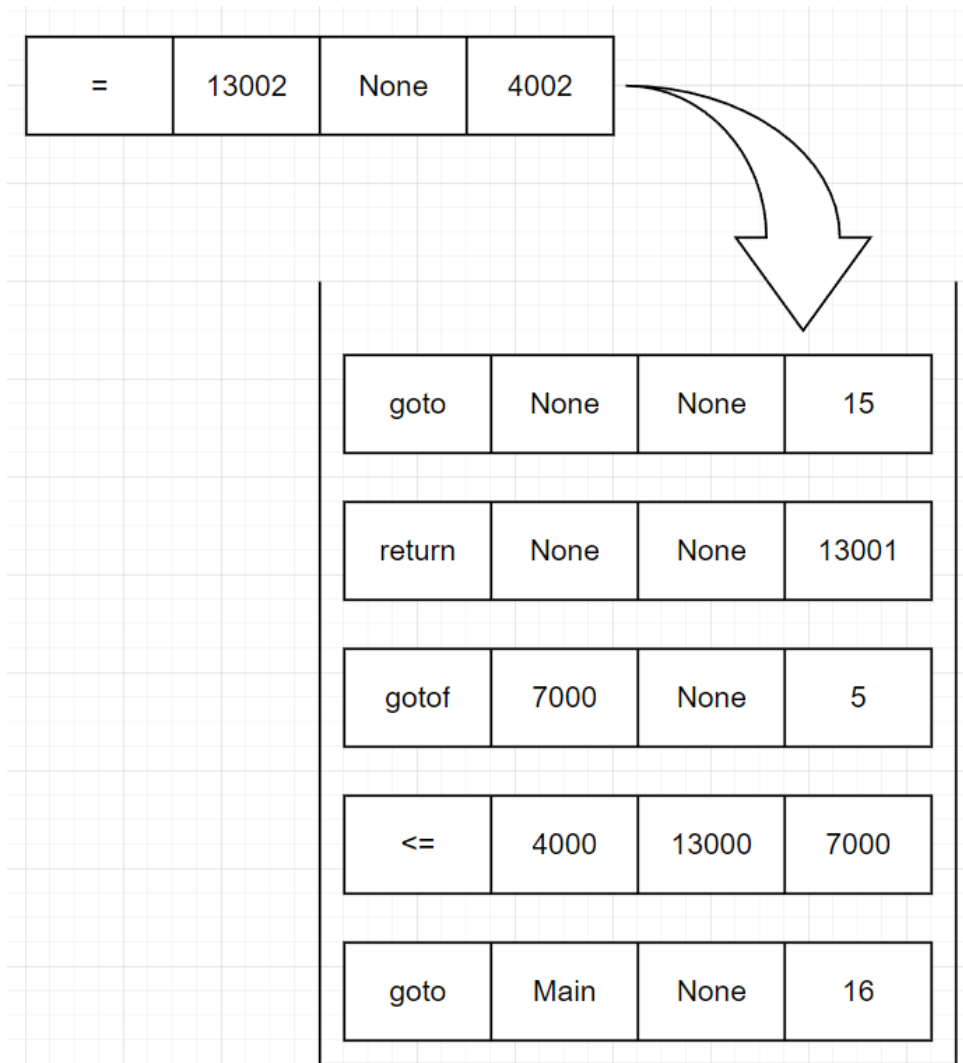
Las tablas de los parámetros y las variables tienen la misma estructura entre ellas. Es una lista de listas que contienen cuatro elementos:

- Name: Nombre de variable/parámetro
- Type: Tipo de dato de variable/parámetro
- Address: Dirección en memoria virtual de variable/parámetro
- Size: Tamaño en caso de que la variable sea arreglo, *None* en el caso contrario



Cuádruplos

Los cuádruplos son simplemente listas de cuatro elementos dentro de una pila implementada con la lista de Python.



Pilas

Se usaron varias pilas para construir el directorio de funcionamiento y para la generación de código. Son las siguientes:

Nombre de pila	Descripción
functionIDs	Guardar los nombres de las funciones que van apareciendo
functionTypes	Guarda el tipo de dato de retorno de las funciones que van apareciendo

functionKinds	Guarda si es programa global o función local de las funciones que van apareciendo
parametersTables	Guarda tablas de parámetros de las funciones que van apareciendo
variablesTables	Guarda tablas de variables de las funciones que van apareciendo
parametersIDsStack	Guarda los parámetros que van apareciendo en la firma de una función
variableIDsStack	Guarda los nombres de las variables que se van declarando
variableSizesStack	Guarda los tamaños de las variables que se van declarando (<i>None</i> si la variable no es arreglo)
variableTypesCountStack	Guarda la cantidad de variables que hay para un determinado tipo de dato
typesStack	Guarda los tipos de datos de las variables que se van declarando
operators	Guarda los operadores que van apareciendo
operands	Guarda los operandos que van apareciendo
types	Guarda los tipos de los operandos que se van guardando
jumps	Guarda los saltos que se deben hacer para los estatutos condicionales
results	Guarda las direcciones de las variables temporales que se van creando
resultsTypes	Guarda los tipos de datos de las variables temporales que se van creando
controlVariables	Guarda las variables que se utilizan para controlar un ciclo <i>for</i>

functionNames	Guarda los nombres de las funciones que se van llamando
functionParametersTables	Guarda las tablas de parámetros de las funciones que se van llamando
functionParameterIndex	Guarda índices que apuntan a la tabla de parámetros de las funciones que se van llamando

Memoria Virtual

La memoria está dividida en cuatro segmentos: memoria global, memoria local, memoria de temporales y memoria de constantes. Cada una de estos espacios se descompone en tres áreas para los tres tipos de datos (*int*, *float* y *char*). Además, el segmento de memoria de temporales tiene otras tres áreas para pointers de cada tipo. Cada área tiene 1000 índices de memoria.

En Python, esta memoria es representada por una lista de 15 diccionarios. Cada diccionario representa una de las 15 áreas de la memoria virtual. Dentro del diccionario, la llave indica la dirección de memoria y el valor es el contenido de esa dirección.

Espacio de memoria	Tipo de dato	Índice inicial	Índice final
Memoria global	int	1000	1999
	float	2000	2999
	char	3000	3999
Memoria local	int	4000	4999
	float	5000	5999
	char	6000	6999
Memoria de temporales	int	7000	7999
	float	8000	8999
	char	9000	9999

	int pointer	10000	10999
	float pointer	11000	11999
	char pointer	12000	12999
Memoria de constantes	int	13000	13999
	float	14000	14999
	char	15000	15999

Descripción de la Máquina Virtual

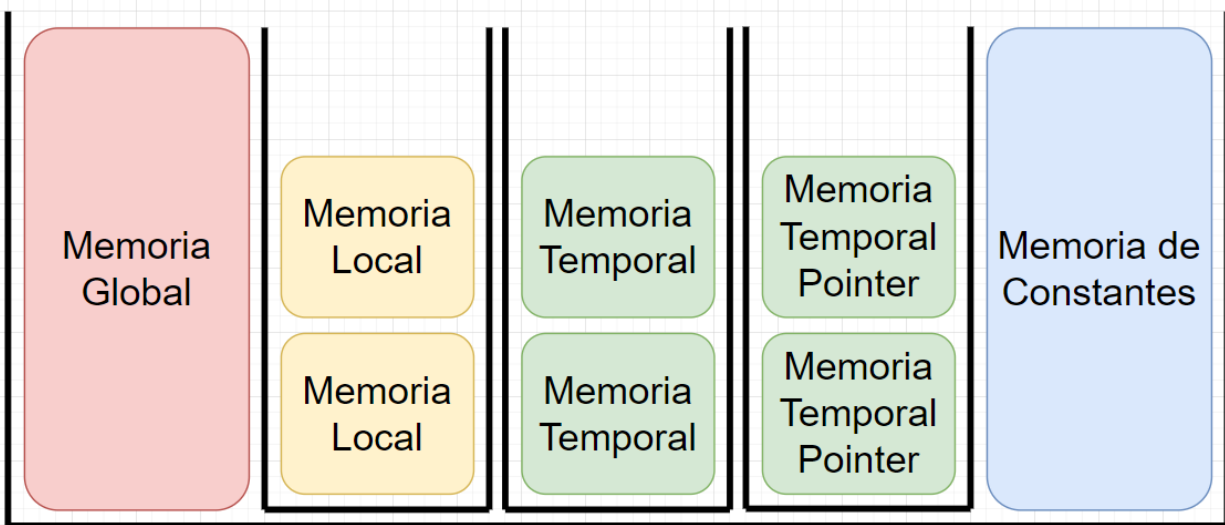
Equipo de cómputo, lenguaje y utilerías

Al igual que el compilador, la máquina virtual está implementada en Python. También usa la librería de JSON para así poder leer el archivo generado por el compilador.

Administración de memoria

La memoria en la máquina virtual es muy similar a la de compilación, pero si tiene algunas en la implementación. En lugar de ser una lista de 15 diccionarios para las áreas, es una lista de 5 listas. Estas 5 listas representan cada espacio de memoria representan los 5 espacios de memoria (global, local, temporal, temporal de pointers y constantes). Cada una de estas se divide en 3 diccionarios que representan las áreas (*int*, *float* y *char*). Pero para poder manejar varias instancias de memoria local y temporal, estas listas no contienen directamente los 3 diccionarios, sino que contienen una lista que funciona como pila donde se pueden agregar varias instancias de estos espacios.

Tiene las mismas direcciones que las establecidas en el compilador, por lo que fácilmente se pueden pasar las constantes encontradas en el compilador a la memoria virtual de la máquina virtual.



Pruebas del Funcionamiento del Lenguaje

Test Case 1: Factorial cíclico

```
> ----- <
      Directorio de Procedimientos
> ----- <
Name:  patito
Type:  void
Kind:  program
Quadruple: 16
Parameters:
Variables:
      Name: a ( int ) -> 1000
-----
Name:  factorial
Type:  int
Kind:  function
Quadruple: 1
Parameters:
      Name: n ( int ) -> 4000
Variables:
```



```

      Name: i ( int ) -> 4001
      Name: result ( int ) -> 4002
-----

> ----- <
                        Cuádruplos
> ----- <
0 : ['goto', 'main', None, 16]
1 : ['<=', 4000, 13000, 7000]
2 : ['gotof', 7000, None, 5]
3 : ['return', None, None, 13001]
4 : ['goto', None, None, 15]
5 : ['=', 13002, None, 4002]
6 : ['=', 13003, None, 4001]
7 : ['+', 4000, 13004, 7001]
8 : ['<', 4001, 7001, 7002]
9 : ['gotof', 7002, None, 14]
10 : ['*', 4002, 4001, 7003]
11 : ['=', 7003, None, 4002]
12 : ['+', 4001, '1', 4001]
13 : ['goto', None, None, 8]
14 : ['return', None, None, 4002]
15 : ['endfunc', None, None, None]
16 : ['era', 'factorial', None, None]
17 : ['param', 13005, None, 'param1']
18 : ['gosub', 'factorial', None, None]
19 : ['=', 'factorial', None, 7000]
20 : ['write', None, None, 7000]
21 : ['end', None, None, None]

```

Test Case 2: Factorial recursivo

```

> ----- <
                        Directorio de Procedimientos
> ----- <
Name: patito
Type: void
Kind: program

```

```

Quadruple: 17
Parameters:
Variables:
    Name: a ( int ) -> 1000
-----
Name: factorial
Type: int
Kind: function
Quadruple: 1
Parameters:
    Name: n ( int ) -> 4000
Variables:
    Name: i ( int ) -> 4001
    Name: result ( int ) -> 4002
-----

> ----- <
                        Cuádruplos
> ----- <
0 : ['goto', 'main', None, 17]
1 : ['<', 4000, 13006, 7000]
2 : ['gotof', 7000, None, 5]
3 : ['return', None, None, 13007]
4 : ['goto', None, None, 16]
5 : ['==', 4000, 13008, 7001]
6 : ['gotof', 7001, None, 9]
7 : ['return', None, None, 13009]
8 : ['goto', None, None, 16]
9 : ['era', 'factorial', None, None]
10 : ['*', 4000, 4000, 7002]
11 : ['-', 7002, 13010, 7003]
12 : ['param', 7003, None, 'param1']
13 : ['gosub', 'factorial', None, None]
14 : ['=', 'factorial', None, 7004]
15 : ['return', None, None, 7004]
16 : ['endfunc', None, None, None]
17 : ['era', 'factorial', None, None]
18 : ['param', 13011, None, 'param1']

```

```

19 : ['gosub', 'factorial', None, None]
20 : ['=', 'factorial', None, 7000]
21 : ['write', None, None, 7000]
22 : ['end', None, None, None]

```

Test Case 3: Fibonacci cíclico

```

> ----- <
                                Directorio de Procedimientos
> ----- <
Name: patito
Type: void
Kind: program
Quadruple: 27
Parameters:
Variables:
    Name: a ( int ) -> 1000
-----
Name: fibonacci
Type: int
Kind: function
Quadruple: 1
Parameters:
    Name: n ( int ) -> 4000
Variables:
    Name: i ( int ) -> 4001
    Name: n1 ( int ) -> 4002
    Name: n2 ( int ) -> 4003
    Name: result ( int ) -> 4004
-----

> ----- <
                                Cuádruplos
> ----- <
0 : ['goto', 'main', None, 27]
1 : ['=', 13012, None, 4002]
2 : ['=', 13013, None, 4003]
3 : ['<=', 4000, 13014, 7000]

```

```

4 : ['goto', 7000, None, 7]
5 : ['return', None, None, 13015]
6 : ['goto', None, None, 26]
7 : ['==', 4000, 13016, 7001]
8 : ['goto', 7001, None, 11]
9 : ['return', None, None, 4002]
10 : ['goto', None, None, 26]
11 : ['==', 4000, 13017, 7002]
12 : ['goto', 7002, None, 15]
13 : ['return', None, None, 4003]
14 : ['goto', None, None, 26]
15 : ['=', 13018, None, 4001]
16 : ['+', 4000, 13019, 7003]
17 : ['<', 4001, 7003, 7004]
18 : ['goto', 7004, None, 25]
19 : ['+', 4002, 4003, 7005]
20 : ['=', 7005, None, 4004]
21 : ['=', 4003, None, 4002]
22 : ['=', 4004, None, 4003]
23 : ['+', 4001, '1', 4001]
24 : ['goto', None, None, 17]
25 : ['return', None, None, 4004]
26 : ['endfunc', None, None, None]
27 : ['era', 'fibonacci', None, None]
28 : ['param', 13020, None, 'param1']
29 : ['gosub', 'fibonacci', None, None]
30 : ['=', 'fibonacci', None, 7000]
31 : ['write', None, None, 7000]
32 : ['end', None, None, None]

```

Test Case 4: Fibonacci recursivo

```

> ----- <
      Directorio de Procedimientos
> ----- <
Name:  patito
Type:  void
Kind:  program
Quadruple:  26

```

Parameters:

Variables:

Name: a (int) -> 1000

Name: fibonacci

Type: int

Kind: function

Quadruple: 1

Parameters:

Name: n (int) -> 4000

Variables:

```
> ----- <
                                Cuádruplos
> ----- <
```

```
0 : ['goto', 'main', None, 26]
1 : ['<=', 4000, 13021, 7000]
2 : ['gotof', 7000, None, 5]
3 : ['return', None, None, 13022]
4 : ['goto', None, None, 25]
5 : ['==', 4000, 13023, 7001]
6 : ['gotof', 7001, None, 9]
7 : ['return', None, None, 13024]
8 : ['goto', None, None, 25]
9 : ['==', 4000, 13025, 7002]
10 : ['gotof', 7002, None, 13]
11 : ['return', None, None, 13026]
12 : ['goto', None, None, 25]
13 : ['era', 'fibonacci', None, None]
14 : ['- ', 4000, 13027, 7003]
15 : ['param', 7003, None, 'param1']
16 : ['gosub', 'fibonacci', None, None]
17 : ['=', 'fibonacci', None, 7004]
18 : ['era', 'fibonacci', None, None]
19 : ['+', 7004, 4000, 7005]
20 : ['- ', 7005, 13028, 7006]
21 : ['param', 7006, None, 'param1']
```

```

22 : ['gosub', 'fibonacci', None, None]
23 : ['=', 'fibonacci', None, 7007]
24 : ['return', None, None, 7007]
25 : ['endfunc', None, None, None]
26 : ['era', 'fibonacci', None, None]
27 : ['param', 13029, None, 'param1']
28 : ['gosub', 'fibonacci', None, None]
29 : ['=', 'fibonacci', None, 7000]
30 : ['write', None, None, 7000]
31 : ['end', None, None, None]

```

Test Case 5: Sort

```

> ----- <
      Directorio de Procedimientos
> ----- <
Name:  MyRlike
Type:  void
Kind:  program
Quadruple:  35
Parameters:
Variables:
      Name:  arreglo [ 10 ] ( int ) -> 1000
      Name:  n ( int ) -> 1010
      Name:  i ( int ) -> 1011
      Name:  j ( int ) -> 1012
-----
Name:  sort
Type:  void
Kind:  function
Quadruple:  1
Parameters:
Variables:
      Name:  temp ( int ) -> 4000
-----

> ----- <
      Cuádruplos

```

```

> ----- <
0 : ['goto', 'main', None, 35]
1 : ['=', 13030, None, 1011]
2 : ['-', 1010, 13031, 7000]
3 : ['<', 1011, 7000, 7001]
4 : ['gotof', 7001, None, 34]
5 : ['=', 13032, None, 1012]
6 : ['-', 1010, 1011, 7002]
7 : ['-', 7002, 13033, 7003]
8 : ['<', 1012, 7003, 7004]
9 : ['gotof', 7004, None, 32]
10 : ['ver', 1012, 13034, 13035]
11 : ['+', 1012, 13036, 10000]
12 : ['+', 1012, 13037, 7005]
13 : ['ver', 7005, 13038, 13039]
14 : ['+', 7005, 13040, 10001]
15 : ['>', 10000, 10001, 7006]
16 : ['gotof', 7006, None, 30]
17 : ['ver', 1012, 13041, 13042]
18 : ['+', 1012, 13043, 10002]
19 : ['=', 10002, None, 4000]
20 : ['ver', 1012, 13044, 13045]
21 : ['+', 1012, 13046, 10003]
22 : ['+', 1012, 13047, 7007]
23 : ['ver', 7007, 13048, 13049]
24 : ['+', 7007, 13050, 10004]
25 : ['=', 10004, None, 10003]
26 : ['+', 1012, 13051, 7008]
27 : ['ver', 7008, 13052, 13053]
28 : ['+', 7008, 13054, 10005]
29 : ['=', 4000, None, 10005]
30 : ['+', 1012, '1', 1012]
31 : ['goto', None, None, 8]
32 : ['+', 1011, '1', 1011]
33 : ['goto', None, None, 3]
34 : ['endfunc', None, None, None]
35 : ['=', 13055, None, 1010]
36 : ['ver', 13056, 13057, 13058]
37 : ['+', 13056, 13059, 10000]

```

```
38 : ['=', 13060, None, 10000]
39 : ['ver', 13061, 13062, 13063]
40 : ['+', 13061, 13064, 10001]
41 : ['=', 13065, None, 10001]
42 : ['ver', 13066, 13067, 13068]
43 : ['+', 13066, 13069, 10002]
44 : ['=', 13070, None, 10002]
45 : ['ver', 13071, 13072, 13073]
46 : ['+', 13071, 13074, 10003]
47 : ['=', 13075, None, 10003]
48 : ['ver', 13076, 13077, 13078]
49 : ['+', 13076, 13079, 10004]
50 : ['=', 13080, None, 10004]
51 : ['ver', 13081, 13082, 13083]
52 : ['+', 13081, 13084, 10005]
53 : ['=', 13085, None, 10005]
54 : ['ver', 13086, 13087, 13088]
55 : ['+', 13086, 13089, 10006]
56 : ['=', 13090, None, 10006]
57 : ['ver', 13091, 13092, 13093]
58 : ['+', 13091, 13094, 10007]
59 : ['=', 13095, None, 10007]
60 : ['ver', 13096, 13097, 13098]
61 : ['+', 13096, 13099, 10008]
62 : ['=', 13100, None, 10008]
63 : ['ver', 13101, 13102, 13103]
64 : ['+', 13101, 13104, 10009]
65 : ['=', 13105, None, 10009]
66 : ['era', 'sort', None, None]
67 : ['gosub', 'sort', None, None]
68 : ['=', 13106, None, 1011]
69 : ['<', 1011, 1010, 7000]
70 : ['gotof', 7000, None, 76]
71 : ['ver', 1011, 13107, 13108]
72 : ['+', 1011, 13109, 10010]
73 : ['write', None, None, 10010]
74 : ['+', 1011, '1', 1011]
75 : ['goto', None, None, 69]
76 : ['end', None, None, None]
```


Test Case 6: Find

```

> ----- <
      Directorio de Procedimientos
> ----- <
Name:  MyRlike
Type:  void
Kind:  program
Quadruple:  14
Parameters:
Variables:
      Name:  arreglo [ 10 ] ( int ) -> 1000
      Name:  n  ( int ) -> 1010
      Name:  i  ( int ) -> 1011
-----
Name:  find
Type:  int
Kind:  function
Quadruple:  1
Parameters:
      Name:  x  ( int ) -> 4000
Variables:
      Name:  index ( int ) -> 4001
-----

> ----- <
      Cuádruplos
> ----- <
0 :  ['goto', 'main', None, 14]
1 :  ['=', 13110, None, 4001]
2 :  ['=', 13111, None, 1011]
3 :  ['<', 1011, 1010, 7000]
4 :  ['gotof', 7000, None, 12]
5 :  ['ver', 1011, 13112, 13113]
6 :  ['+', 1011, 13114, 10000]
7 :  ['==', 10000, 4000, 7001]
8 :  ['gotof', 7001, None, 10]
9 :  ['=', 1011, None, 4001]

```

```
10 : ['+', 1011, '1', 1011]
11 : ['goto', None, None, 3]
12 : ['return', None, None, 4001]
13 : ['endfunc', None, None, None]
14 : ['=', 13115, None, 1010]
15 : ['ver', 13116, 13117, 13118]
16 : ['+', 13116, 13119, 10000]
17 : ['=', 13120, None, 10000]
18 : ['ver', 13121, 13122, 13123]
19 : ['+', 13121, 13124, 10001]
20 : ['=', 13125, None, 10001]
21 : ['ver', 13126, 13127, 13128]
22 : ['+', 13126, 13129, 10002]
23 : ['=', 13130, None, 10002]
24 : ['ver', 13131, 13132, 13133]
25 : ['+', 13131, 13134, 10003]
26 : ['=', 13135, None, 10003]
27 : ['ver', 13136, 13137, 13138]
28 : ['+', 13136, 13139, 10004]
29 : ['=', 13140, None, 10004]
30 : ['ver', 13141, 13142, 13143]
31 : ['+', 13141, 13144, 10005]
32 : ['=', 13145, None, 10005]
33 : ['ver', 13146, 13147, 13148]
34 : ['+', 13146, 13149, 10006]
35 : ['=', 13150, None, 10006]
36 : ['ver', 13151, 13152, 13153]
37 : ['+', 13151, 13154, 10007]
38 : ['=', 13155, None, 10007]
39 : ['ver', 13156, 13157, 13158]
40 : ['+', 13156, 13159, 10008]
41 : ['=', 13160, None, 10008]
42 : ['ver', 13161, 13162, 13163]
43 : ['+', 13161, 13164, 10009]
44 : ['=', 13165, None, 10009]
45 : ['era', 'find', None, None]
46 : ['param', 13166, None, 'param1']
47 : ['gosub', 'find', None, None]
48 : ['=', 'find', None, 7000]
```

```

49 : ['write', None, None, 7000]
50 : ['end', None, None, None]

```

Test Case 7: Operaciones sobre arreglos

```

> ----- <
      Directorio de Procedimientos
> ----- <
Name:  MyRlike
Type:  void
Kind:  program
Quadruple:  15
Parameters:
Variables:
    Name:  arreglo1 [ 5 ] ( int ) -> 1000
    Name:  arreglo2 [ 5 ] ( int ) -> 1005
    Name:  arreglo3 [ 5 ] ( int ) -> 1010
    Name:  n  ( int ) -> 1015
    Name:  i  ( int ) -> 1016
-----
Name:  multiply
Type:  void
Kind:  function
Quadruple:  1
Parameters:
Variables:
-----

> ----- <
      Cuádruplos
> ----- <
 0 : ['goto', 'main', None, 15]
 1 : ['=', 13167, None, 1016]
 2 : ['<', 1016, 1015, 7000]
 3 : ['gotof', 7000, None, 14]
 4 : ['ver', 1016, 13168, 13169]
 5 : ['+', 1016, 13170, 10000]
 6 : ['ver', 1016, 13171, 13172]

```

```
7 : ['+', 1016, 13173, 10001]
8 : ['*', 10001, 1016, 7001]
9 : ['ver', 7001, 13174, 13175]
10 : ['+', 7001, 13176, 10002]
11 : ['=', 10002, None, 10000]
12 : ['+', 1016, '1', 1016]
13 : ['goto', None, None, 2]
14 : ['endfunc', None, None, None]
15 : ['=', 13177, None, 1015]
16 : ['ver', 13178, 13179, 13180]
17 : ['+', 13178, 13181, 10000]
18 : ['=', 13182, None, 10000]
19 : ['ver', 13183, 13184, 13185]
20 : ['+', 13183, 13186, 10001]
21 : ['=', 13187, None, 10001]
22 : ['ver', 13188, 13189, 13190]
23 : ['+', 13188, 13191, 10002]
24 : ['=', 13192, None, 10002]
25 : ['ver', 13193, 13194, 13195]
26 : ['+', 13193, 13196, 10003]
27 : ['=', 13197, None, 10003]
28 : ['ver', 13198, 13199, 13200]
29 : ['+', 13198, 13201, 10004]
30 : ['=', 13202, None, 10004]
31 : ['ver', 13203, 13204, 13205]
32 : ['+', 13203, 13206, 10005]
33 : ['=', 13207, None, 10005]
34 : ['ver', 13208, 13209, 13210]
35 : ['+', 13208, 13211, 10006]
36 : ['=', 13212, None, 10006]
37 : ['ver', 13213, 13214, 13215]
38 : ['+', 13213, 13216, 10007]
39 : ['=', 13217, None, 10007]
40 : ['ver', 13218, 13219, 13220]
41 : ['+', 13218, 13221, 10008]
42 : ['=', 13222, None, 10008]
43 : ['ver', 13223, 13224, 13225]
44 : ['+', 13223, 13226, 10009]
45 : ['=', 13227, None, 10009]
```

```
46 : ['era', 'multiply', None, None]
47 : ['gosub', 'multiply', None, None]
48 : ['=', 13228, None, 1016]
49 : ['<', 1016, 1015, 7000]
50 : ['gotof', 7000, None, 56]
51 : ['ver', 1016, 13229, 13230]
52 : ['+', 1016, 13231, 10010]
53 : ['write', None, None, 10010]
54 : ['+', 1016, '1', 1016]
55 : ['goto', None, None, 49]
56 : ['end', None, None, None]
```