Tecnológico de Monterrey

Diseño de Compiladores
Alebrije

Profesores:
Elda G. Quiroga, M.C.
Dr. Héctor Ceballos, PhD

Gerardo Galan Garzafox A00821196

24 de Noviembre del 2021

Indice

1. DESCRIPCIÓN DEL PROYECTO

1.1. Propósito y Alcance del Proyecto

El propósito de este proyecto es el de realizar un programa capaz de compilar y ejecutar a un lenguaje programación denominado alebrije que es basado en R. Este debe de ser orientado a jóvenes que buscan aprender los fundamentos de la programación, a través del manejo y manipulación de conjuntos de datos simples para realizar análisis estadístico básico.El alcance del proyecto convertir el lenguaje en tokens(léxico), verificar el orden lógico de estos(sintaxis) y el sentido de este orden(semántica). El alcance del lenguaje debe de incluir: expresiones aritméticas/lógicas y relacionales; estatutos de interacción (entrada/salida); estatutos de flujo(ciclo, decisiones); elementos de cambio de contexto(funciones parametrizables); manejo de elementos no atómicos(arreglos) y funciones predefinidas de estadística(Media, Moda, Varianza, Regresión Simple, PlotXY, etc...)

1.2. Análisis de Requerimientos y Test Cases

REQUISITOS

ID	R1	**Prioridad:**	Alta
Nombre	Expresiones aritméticas/lógicas y relacionales	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de ser capaz de realizar expresiones - aritméticas(*, /, ^, +, -, %) - lógicas(<, >, >=, <=, ==, !=) - relacionales (and, or) El orden de estas es operando izquierdo, operador, operando derecho. Se pueden utilizar variables constantes de los tipos de datos disponibles, llamadas a funciones, funciones predefinidas o variables para realizar estas operaciones.		
Pre-Condiciones:	Las funciones y variables deben de estar previamente definidas, además las variables deben de tener un valor asignado .		
Post-Condiciones:	Los operandos deben de tener tipos de datos compatibles con sigo mismos y con el operador.		

ID	R2	**Prioridad:**	Alta
Nombre	Asignación de variables (=,+=,-=,*=,/=)	**Creado por:**	Gerardo
Descripción:	Dentro del lenguaje es posible asignarle el valor resultante de una expresión a una variable ya declarada.		
Pre-Condiciones:	La variable debe de estar previamente definida.		
Post-Condiciones:	La variable debe de recordar su último valor asignado. El tipo de datos a asignar debe de ser compatible con el tipo de dato de la variable.		

ID	R3	**Prioridad:**	Baja
Nombre	Entrada/Lectura/Input (read)	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar la capacidad de aceptar entradas de consola por parte del usuario y asignarlas a la variable.		
Pre-Condiciones:	La variable a leer debe de estar previamente definida.		
Post-Condiciones:	La variable debe de recordar su nuevo valor asignado. El tipo de dato asignado a la variable debe de ser compatible con el tipo de datos de la variable.		

ID	R4	**Prioridad:**	Baja
Nombre	Salida/Escritura/Output (write)	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de ser capaz de hacer impresiones en consola de expresiones y string constantes.		
Pre-Condiciones:	Las variables a escribir deben de estar previamente definidas.		
Post-Condiciones:	NA		

ID	R5	**Prioridad:**	Alta
Nombre	Estatuto de control, decisión condicional (if, else)	**Creado por:**	Gerardo
Descripción:	El usuario del lenguaje es capaz de definir el flujo del programa con estatutos condicionales if, else.		
Pre-Condiciones:	El resultado de la expresión del if debe de ser tipo booleano..		
Post-Condiciones:	No se pueden declarar nuevas variables dentro de los bloques de decisión. Es posible no tener un else después del if.		

ID	R6	**Prioridad:**	Alta
Nombre	Estatuto de control, ciclo condicional (while)	**Creado por:**	Gerardo
Descripción:	El lenguaje soporta estatutos de control de tipo bucle condicional como el while, el cual es capaz de repetir estatutos dentro de sí mismo incluyendo otros ciclos.		
Pre-Condiciones:	La expresión dentro del while debe de dar como resultado un valor booleano ya sea falso o verdadero.		
Post-Condiciones:	No se pueden declarar nuevas variables dentro del bloque de decisión del while.		

ID	R7	**Prioridad:**	Media
Nombre	Estatuto de control, ciclo de rango (for)	**Creado por:**	Gerardo
Descripción:	El lenguaje soporta estatutos de control de tipo bucle de rango como el for to, el cual es capaz de repetir estatutos dentro de sí mismo incluyendo otros ciclos.		
Pre-Condiciones:	Las 2 expresiones dentro del ciclo for deben de dar como resultado un valor entero.		
Post-Condiciones:	No se pueden declarar nuevas variables dentro del bloque de decisión del while.		

ID	R8	**Prioridad:**	Alta
Nombre	Definición de elementos de cambio de contexto (funciones parametrizables)	**Creado por:**	Gerardo
Descripción:	El lenguaje cambia de contexto por medio de la declaración y llamada a funciones.		
Pre-Condiciones:	No se le puede llamar a una función antes de ser declarada.		
Post-Condiciones:	Las funciones Void no pueden tener retornar valores. El número de argumentos de una llamada a una función debe de ser igual al número de parámetros de esa función.		

ID	R9	**Prioridad:**	Bajo
Nombre	Definición de elementos de cambio de contexto sin valor de retorno(funciones void)	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar funciones Void.		
Pre-Condiciones:	La función debe de ser declarada como tipo de dato void, en caso de ser definida por el usuario.		
Post-Condiciones:	Las funciones Void no pueden tener retornar valores. El número de argumentos de una llamada a una función debe de ser igual al número de parámetros de esa función.		

ID	R10	**Prioridad:**	Media
Nombre	Ejecución de funciones parametrizables recursivas	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar funciones recursión.		
Pre-Condiciones:	La función debe de estar declarada.		
Post-Condiciones:	La función debe de tener alguna manera proporcionada por el usuario de detener el ciclo de recursión.		

ID	R11	**Prioridad:**	Alto
Nombre	Manejo de arreglos	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar el manejo de arreglos.		
Pre-Condiciones:	El arreglo debe de estar declarado antes de llamarse. El arreglo debe de tener una dimensión.		
Post-Condiciones:	Se debe verificar que el inicio se encuentre dentro de los límites del arreglo.		

ID	R12	**Prioridad:**	Media
Nombre	Ejecución de llamadas de funciones parametrizadas desde los estatutos de otras funciones parametrizadas	**Creado por:**	Gerardo
Descripción:	Es posible hacer llamadas a funciones desde el interior de una función. Esto puede ser una llamada a sí misma(recursión) o a una función completamente distinta.		
Pre-Condiciones:	Las funciones deben de estar declaradas antes de ser llamadas.		
Post-Condiciones:	Las funciones de tipo void no pueden regresar un valor.		

ID	R13	**Prioridad:**	Media
Nombre	Funciones Estadísticas(Media, Moda, Varianza, Regresión Simple, PlotXY)	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar el uso de funciones predefinidas para el uso de estadística básica similar al lenguaje de programación R.		
Pre-Condiciones:	Se le deben de pasar el número correcto de argumentos a cada función.		
Post-Condiciones:	Las funciones regression y plotxy deben de ser tratadas como funciones de tipo void. No se les puede asignar su resultado (porque no devuelve ninguno) a una variable.		

TEST CASES

ID	T1	**Prioridad:**	Alta
Nombre	Fibonacci	**Creado por:**	Gerardo
Descripción:	Programa que calcula la secuencia de números fibonacci por medio de una función auxiliar y la recursión. La secuencia de número fibonacci consiste en sumar los 2 últimos números para generar al siguiente.		
Requisitos aplicables:	R1,R2,R4,R5, R9, R10, R12		
Entradas de usuario:	NA		

ID	T2	**Prioridad:**	Alta
Nombre	Busqueda Binaria	**Creado por:**	Gerardo
Descripción:	Este programa consiste en recibir un número entero del usuario el cual va a ser buscado en una lista ordenada de manera ascendente utilizando la búsqueda binaria.		
Requisitos aplicables:	R1,R2,R3,R4,R5, R7,R8, R10, R11, R12		
Entradas de usuario:	Número entero a buscar		

ID	T3	**Prioridad:**	Media
Nombre	Bubble Sort	**Creado por:**	Gerardo
Descripción:	Este es un programa que ordena una lista desordenada utilizando el algoritmo bubble sort.		
Requisitos aplicables:	R1, R2, R4, R5, R6, R9, R11		
Entradas de usuario:	NA		

ID	T4	**Prioridad:**	Alta
Nombre	Quick Sort	**Creado por:**	Gerardo
Descripción:	Este es un programa que ordena una lista desordenada utilizando el algoritmo quicksort.		
Requisitos aplicables:	R1, R2, R4, R5, R7, R8, R9, R10, R11, R12		
Entradas de usuario:	NA		

ID	T5	**Prioridad:**	Alta
Nombre	Factorial recursivo	**Creado por:**	Gerardo
Descripción:	Este es un programa que realiza el cálculo del factorial de un número de manera recursiva.		
Requisitos aplicables:	R1, R2,R4, R5, R8, R10		
Entradas de usuario:	NA		

ID	T6	**Prioridad:**	Alta
Nombre	Factorial iterativo	**Creado por:**	Gerardo
Descripción:	Este es un programa que realiza el cálculo del factorial de un número de manera iterativa.		
Requisitos aplicables:	R1, R2,R4, R5, R6, R8		
Entradas de usuario:	NA		

ID	T7	**Prioridad:**	Media
Nombre	Números Primos	**Creado por:**	Gerardo
Descripción:	Este programa encuentra todos los números primos menores a 100.		
Requisitos aplicables:	R1, R2, R4, R5, R7, R11		
Entradas de usuario:	NA		

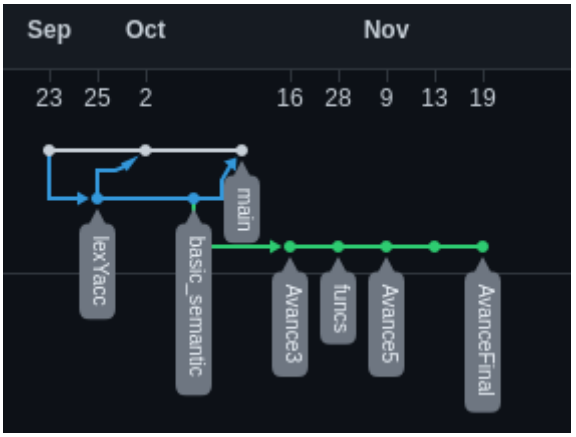
ID	T8	**Prioridad:**	Alta
Nombre	Estadistica	**Creado por:**	Gerardo
Descripción:	Este programa utiliza todas las funciones predefinidas para cálculos estadísticos: mean, median, mode, variance, plotxy, recursión.		
Requisitos aplicables:	R1, R2, R4, R5, R6, R11, R13		
Entradas de usuario:	NA		

ID	T9	**Prioridad:**	Baja
Nombre	Suma de arreglos	**Creado por:**	Gerardo
Descripción:	Este programa guarda la suma de 2 arreglos en un tercer arreglo después lo imprime en pantalla.		
Requisitos aplicables:	R1, R2, R4, R7, R9, R11		
Entradas de usuario:	NA		

ID	T10	**Prioridad:**	Baja
Nombre	Regla trapezoidal	**Creado por:**	Gerardo
Descripción:	Este programa toma como entrada 3 números del usuario y luego calcula la integral de una función con el algoritmo trapezoidal.		
Requisitos aplicables:	R1, R2, R3, R4, R7, R8, R12		
Entradas de usuario:	NA		

1.3. Descripción del PROCESO

Durante el proceso de desarrollo de este proyecto utilice la herramienta de control de versiones git y el servicio de repositorios remotos de github. En mi maquina local utilice VS Code y una terminal de linux para el desarrollo. Intente ir con acorde al calendario de entregas, hubo adelantos y retrasos.



Grafo de commits

Bitacora Semanal

Semana #	**Entregue**
1	Lexer y Parser inicial.
2	Directorio de procedimientos y tabla de variables.
3	Puntos neurálgicos para la aritmética básica y decisiones incompletas.
4	Puntos neurálgicos para ciclos, mapa de memoria.
5	Máquina virtual inicial, direcciones de memoria, ejecución de condicionales y ciclos en máquina virtual.
6	Cambio en manejo de memoria,ejecución de arreglos y funciones en máquina virtual.

7	Documentación 1era version, funciones predefinidas, pruebas
Final	

Compromisos

2. DESCRIPCIÓN DEL LENGUAJE

2.1. Nombre del lenguaje

Alebrije

2.2. Características del lenguaje

Alebrije es un lenguaje de programación básico basado en R para jóvenes que quieren aprender los fundamentos de la programación y análisis estadístico básico. Permite realizar operaciones de aritmética básica, asignación a variables, control de flujo con condicionales, bucles/ciclos, programación modular con funciones, y el uso de arreglos. Además de esto cuenta con facilidades por medio de funciones predefinidas para realizar cálculos estadísticos básicos como encontrar la media, varianza, la moda, promedio, graficación sobre plano cartesiano y regresiones lineales simples.

2.3. Listado de Errores

Compilacion	
1-	Illegal character “{}”
2-	Syntax error found at line {}
3-	Missing type for function: "{}"
4-	The number of arguments doesn't match the number of parameters
5-	Undefined function: "{}"
6-	No file name was provided
7-	EOF Error
8-	Error while trying operation: {type} {oper} {type}
9-	void functions should not return a value
10-	A global variable with that name already exists
11-	Repeated variable name:{var_name} in {func_name}
12-	A procedure with the name "{proc_name}" already exists

13-	A variable with the name "{var_name}" doesn`t exist
14-	No dimension was found for the array with name: "{var_name}"
15-	Procedures main or program can't have a return statement
16-	Array index must be a integer value
17-	Address has no datatype
18-	Function {func_name} only accepts arrays of int's or float's
19-	Function argument doesn’t exist
20-	function {func_name} returns a non existant value
21-	Trying to read non-existent variable
22-	Trying to write non existant value

Ejecución	
1-	CHAR data types can’t be longer than 1 character
2-	Can’t assign {value} to datatype {datatype}
3-	index "{}" of array is out of bounds
4-	char and string values can't be assigned to {datatype}
5-	Could not assign {value} to datatype {datatype}
6-	Value '{value}' isn't of type {datatype}
7-	value '{value}' can't be assigned to a {datatype}
8-	Can't divide by zero

3. DESCRIPCIÓN DEL COMPILADOR

3.1. Descripción técnica.

Sistema Operativo: Manjaro Linux x86_64
Laptop: HP Pavilion Laptop 15-cw1xxx
Kernel: 5.10.70-1-MANJARO
Shell: zsh 5.8
CPU: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx (8) @ 2.300GHz
GPU: AMD ATI 03:00.0 Picasso
Memoria: 7204MiB / 13964MiB

Lenguaje: Python 3.9.7
Utilerias especiales:

- ply.lex
- ply.yacc
- sys
- json
- tabulate

3.2. Descripción del Análisis de Léxico

Tokens

#	Token	Código asociado	Expresion Regular
1	PROGRAM	programa	program\b
2	MAIN	Funcion principal	main\b
3	VARS	variables	vars\b
4	INT	entreo	int\b
5	FLOAT	flotante	float\b
6	BOOL	booleano	bool\b
7	CHAR	caracter	char\b
8	STRING	string	string\b
9	FUNCTION	funcion	function\b
10	RETURN	retorno	return \b
11	READ	leer	read\b
12	WRITE	escribir	write\b
13	IF	si	if\b
14	ELSE	sino	else\b
15	WHILE	mientras	while\b
16	FOR	para	for\b
17	TO	hasta	to\b
18	VOID	vacio	void\b
19	AND	y	and\b

20	OR	o	or\b
21	MEDIAN	media	median\b
22	MODE	moda	mode\b
23	MEAN	promedio	mean\b
24	VARIANCE	varianca	variance\b
25	REGRESSION	regression	regression\b
26	PLOTXY	Graficar x y	plotxy\b
27	MAX	maximo	max\b
28	MIN	minimo	min\b
29	LEN	longitud	len\b
30	LESS	Menos que	<
31	GREATER	Mayor que	>
32	LESS_EQ	Menor o igual que	<=
33	GREATER_EQ	Mayor o igual que	>=
34	EQUIVALENT	Equivalnete a	==
35	DIFFERENT	Differente a	!=
36	EQUAL	igual	=
37	MULT	multiplicacion	*
38	DIV	division	/
39	PLUS	suma	+
40	MINUS	resta	-
41	REMAINDER	resto	%
42	EXP	exponencial	^
43	MULT_EQ	Multiplicar e igualar	*=
44	DIV_EQ	Dividir e igualar	/=
45	PLUS_EQ	Sumar e igualar	+=
46	MINUS_EQ	Restar e igualar	-=
47	L_BRACE	Llave izquierda	{
48	R_BRACE	Llave derecha	}

49	L_BRACKET	Corchete izquierdo	[
50	R_BRACKET	Corchete derecho	]
51	L_PAR	Parentheses izquierdo	(
52	R_PAR	Parenthesis derecho	)
53	COLON	2 puntos	:
54	SEMICOLON	Punto y coma	;
55	COMMA	coma	,
56	ID	identificador	[a-ZA-Z][a-zA-Z_0-9]*
57	CTE_INT	Constantine entera	-?\d+
58	CTE_FLOAT	Constantine flotante	-?\d+\.\d+
59	CTE_BOOL	Constate booleana	(True| False | true | false)
60	CTE_CHAR	Constaten caracter	(\".\" | \'.\')
61	CTE_STRING	Constante string	(\".+\" | \'.+\')

3.3. Descripción del Análisis de Sintaxis.

Gramatica Formal

<PROGRAM>
A → program id ; B
B → VARS C | C
C → FUNCTION C | MAIN

<VARS>
A → vars { B }
B → TYPE : C ; | TYPE : C ; B
C → D | D , C
D → ID | ID [cte_int]

<FUNCTION>
A → function FUNC_TYPE id (B)
VBLOCK
B → PARAMS
→ epsilon

<MAIN>
A → main ( ) VBLOCK

<TYPE>
A → int
→ float

<VAR>
A → id
→ id [EXP]

→ char
→ bool
→ string

<FUNC_TYPE>
A → void | TYPE

<PARAMS>
A → TYPE : id
→ TYPE : id , PARAMS
→epsilon

<BLOCK>
A → { B }
B → STATEMENTS B
→ epsilon

<VBLOCK>
A → { VARS B }
→ BLOCK
B → STATEMENTS B
→ epsilon

<STATEMENTS>
A → ASSIGN
→ CONDICIONAL
→ READ
→ WRITE
→ LOOP_COND
→ LOOP_RANGE
→ RETURN
→ VOID_FUNC

<VOID_FUNC>
A → FUNC_CALL ;
→ plotxy (id, id) ;
→ regression (id , id) ;

<ASSIGN>
A → VAR OPER_ASSIGN EXPRESSION ;

<CONDICIONAL>
A → if (EXPRESSION) BLOCK B
B → else BLOCK
→epsilon

<READ>
A → read (B) ;
B → VAR
→ VAR , B

<WRITE>
A → write (B) ;
B → EXPRESSION C
→ cte.string C
C → , B
→ epsilon

<LOOP_COND>

<LOOP_RANGE>

A → while (EXPRESSION) BLOCK

A → for VAR = EXP to EXP BLOCK

<RETURN>
A → return (EXP) ;

<FUNC_CALL>
A → id (ARGUMENTS)
B →ARGUMENTS
→epsilon

<ARGUMENTS>
A → EXP B
B → , A
→ epsilon

<DEF_FUNC>
A →B (id)
B →min
→ max
→ mean
→ median
→ mode
→ variance
→ len

<OPER_ASSIGN>
A → =
→+=
→ -=
→ *=
→ /=

<EXPRESSION>
A → LOGIC B
B → or A
→and A
→ epsilon

<LOGIC>
A → EXP B
B → < EXP
→ > EXP
→ <= EXP
→>= EXP
→== EXP
→ != EXP
→epsilon

<EXP>
A → TERM B
B → - A
→ + A
→ epsilon

<TERM>

<EXPONENT>

A → EXPONENT B
B → * A
→ / A
→% A
→ epsilon

A → FACTOR B
B → ^ A
→epsilon

<FACTOR>
A → (EXPRESSION)
→ -VAR_CTE
→ VAR_CTE

<VAR_CTE>
A → VAR
→FUNC_CALL
→DEF_FUNC
→ cte.int
→ cte.float
→ cte.char
→ cte.string
→ cte.bool

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

- o Diagramas de Sintaxis con las acciones correspondientes marcadas sobre ellos (puntos neurálgicos).
- o Breve descripción de cada una de las acciones semánticas y de generación de código (no más de 2 líneas).

Codigos de Operacion

GOTO	>	=	REGRESSION
GOTOF	and	MAX	ERA

WRITE	or	MIN	PARAM
READ	%	MEAN	GOSUB
==	^	MEDIAN	RET
!=	*	MODE	ENDFUNC
>=	/	VARIANCE	VER
<=	+	LEN	[+]
<	-	PLOTXY	ENDPROG

Direcciones virtuales asociadas a los elementos del código

Globales	
int	1000-1999
float	2000-2999
bool	3000-3999
char	4000-4999
string	5000-5999

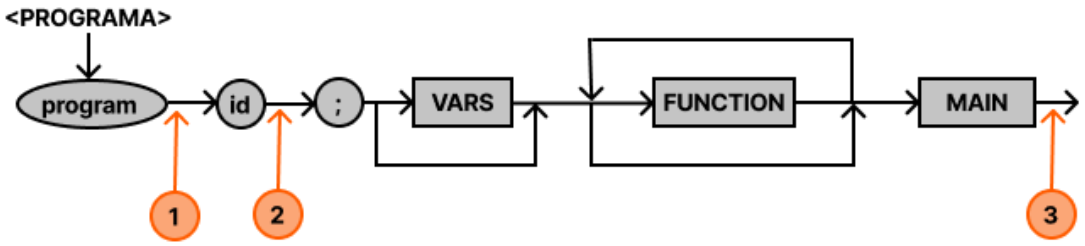
Locales	
int	6000-6999
float	7000-7999
bool	8000-8999
char	9000-9999
string	10000-10999

Temporales	
int	11000-11999
float	12000-12999
bool	13000-13999
char	14000-14999
string	15000-15999

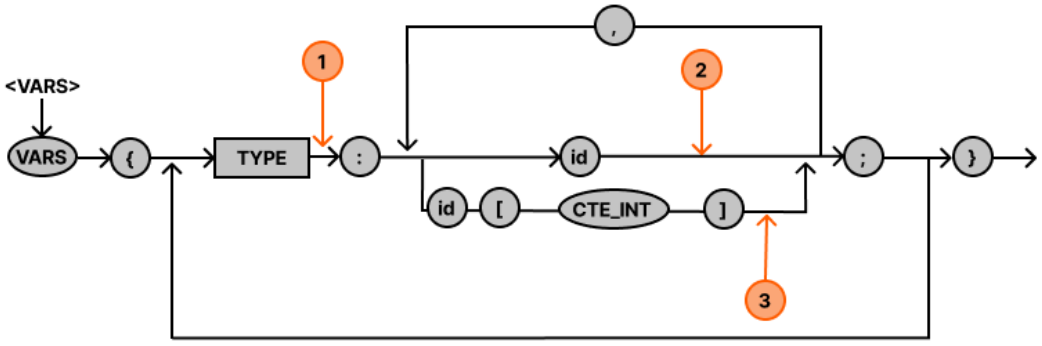
Constantes	
int	16000-16999
float	17000-17999
bool	18000-18999
char	19000-19999
string	20000-20999

Apuntadores
21000...

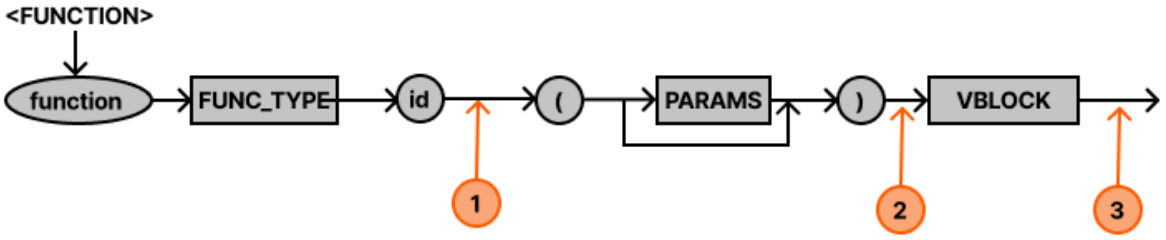
Diagrama de Sintaxis y acciones semánticas



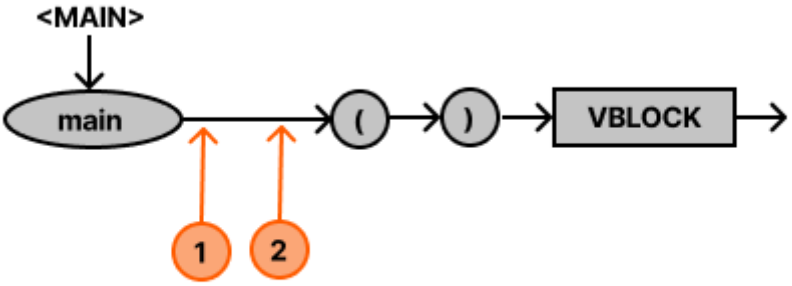
PROGRAM	
#	**Accion semantica**
1	Cambiar procedimiento actual a ‘program’.
2	-Generar cuádruplo GOTO. -Hacer push() de cuádruplo actual a la pila de saltos.
3	Generar cuádruplo PROGEND.



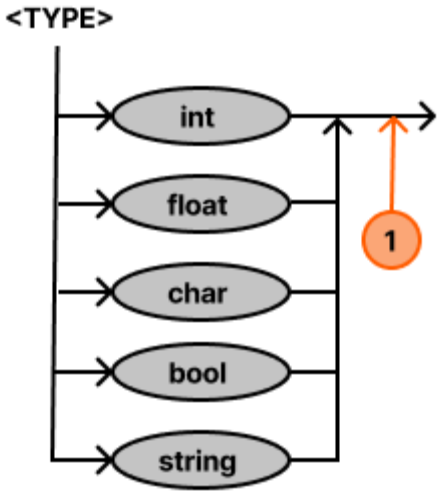
VARS	
#	**Accion semantica**
1	Cambiar el tipo de dato actual.
2	Añadir variable a la tabla de variables de la función actual.
3	Añadir id del arreglo y su dimensión a la tabla de variables de la función actual.



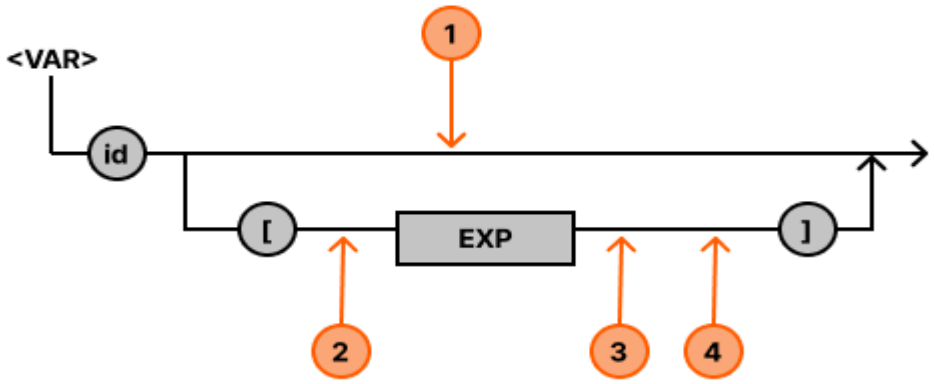
FUNCTION	
#	**Accion semantica**
1	Cambiar procedimiento actual a id de función.
2	Añadir cuádruplo donde empieza la función al directorio de procedimientos.
3	Generar cuádruplo FUNCEND.



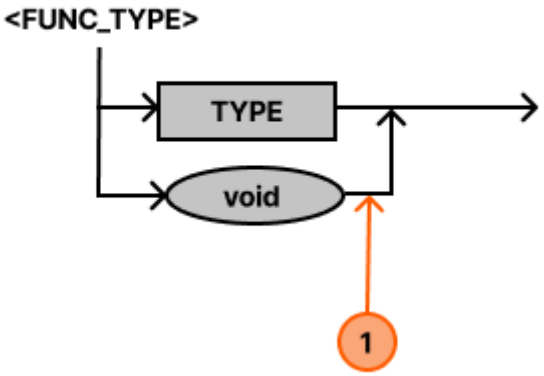
MAIN	
#	**Accion semantica**
1	Cambiar el procedimiento actual a ‘main’.
2	-Hacer pop() de la pila de saltos. -Agregar el número de cuádruplo actual al GOTO de programa.



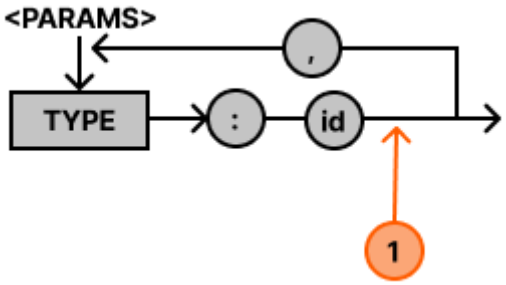
TYPE	
#	**Accion semantica**
1	Hacer push() del tipo de dato a la pila de tipos de datos.



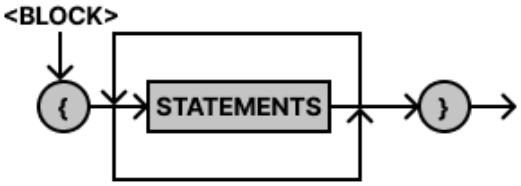
VAR	
#	**Accion semantica**
1	Hacer push() de id a la pila de operandos.
2	Hacer push() de ‘stop’ a la pila de operadores.
3	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
4	-Hacer pop() de la pila de operandos para obtener el índice del arreglo. -Generar cuádruplo de verificación con índice, limite inferior, limite superior. -Generar cuádruplo de la suma entre índice y dirección base en un apuntador.

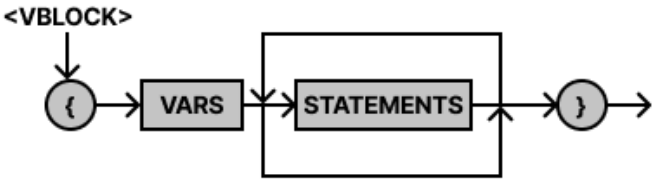


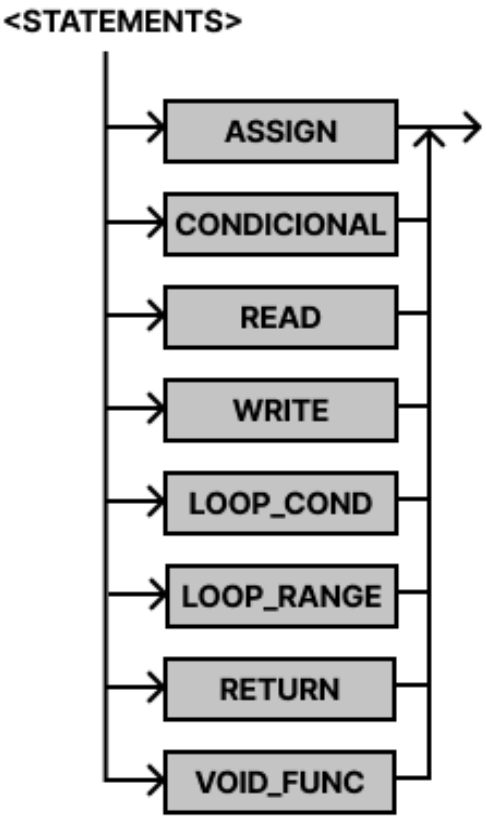
FUNC_TYPE	
#	**Accion semantica**
1	Hacer push() del tipo de dato a la pila de tipos de datos.

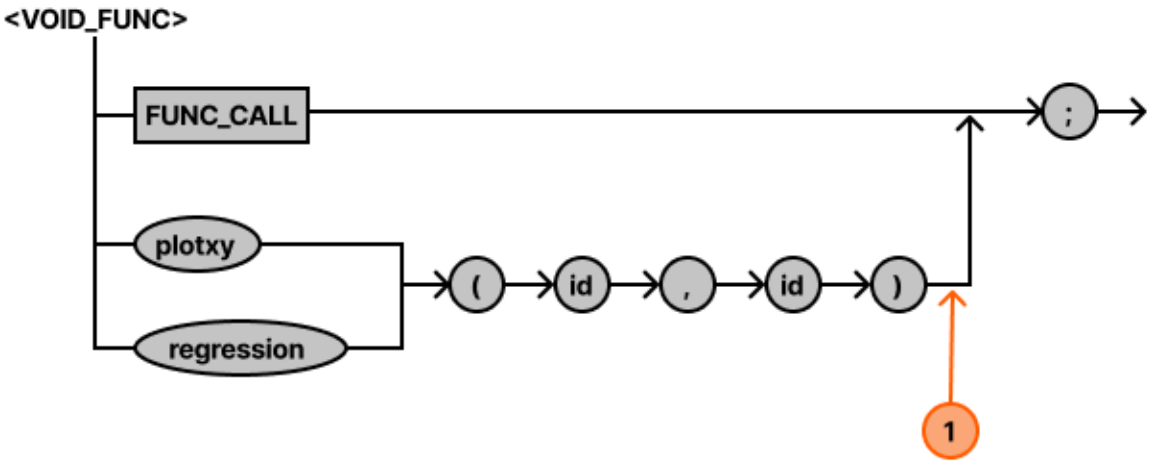


PARAMS	
#	**Accion semantica**
1	- Agrega id y tipo del parámetro a la tabla de variables de la función actual. - Agrega tipo al vector de parámetros de la función.

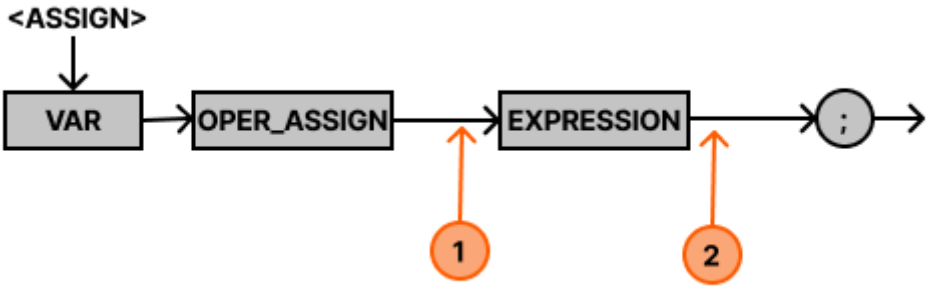




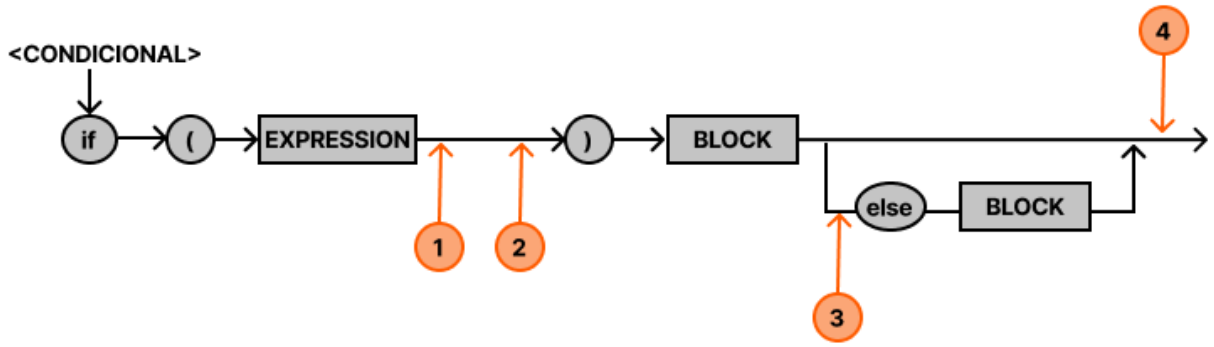




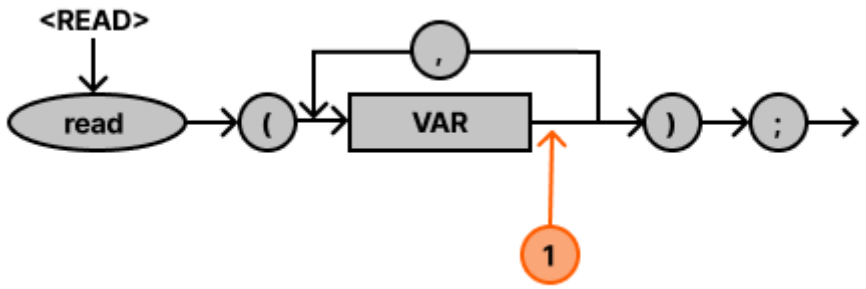
VOID_FUNC	
#	**Accion semantica**
1	- Generar cuádruplos de PLOTXY o REGRESSION.



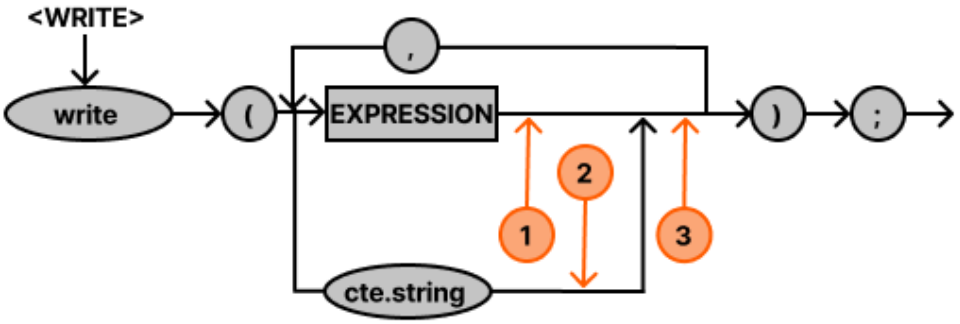
ASSIGN	
#	**Accion semantica**
1	Hacer push() del operador a pila de operadores.
2	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.



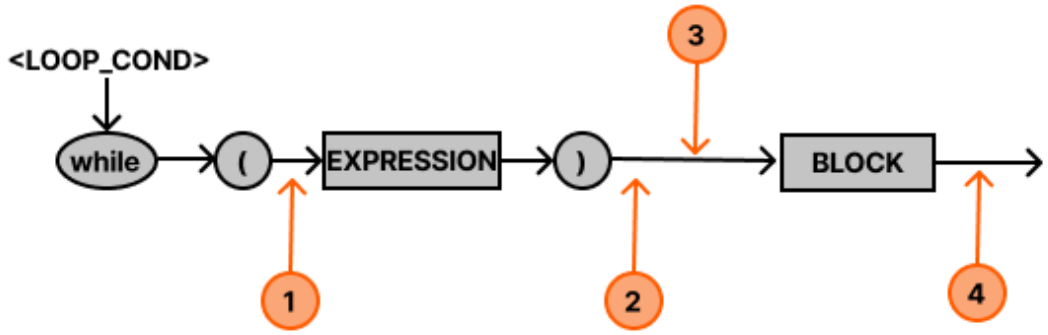
CONDICIONAL	
#	**Accion semantica**
1	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
2	- Hacer pop() de la pila de operandos. - Generar cuádruplo de GOTOF.
3	- Hacer pop() de pila de saltos y agregar número de cuádruplo actual al GOTOF del if. - Agregar número de cuádruplo actual a la pila de saltos. - Generar cuádruplo GOTO.
4	Hacer pop() de pila de saltos y agregar número de cuádruplo actual al GOTOF del if o al GOTO del else.



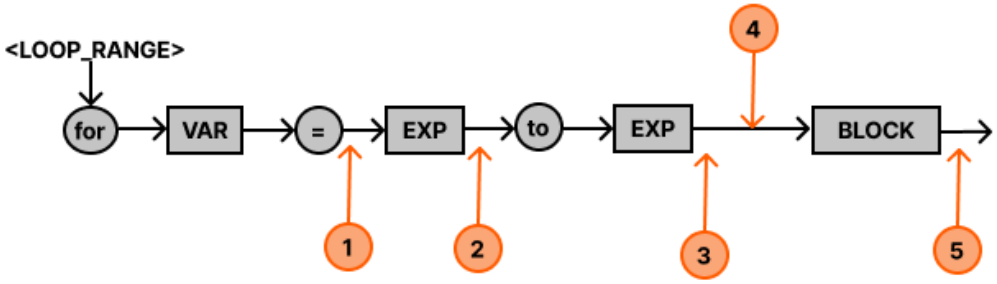
READ	
#	**Accion semantica**
1	- Hacer pop() de la pila de operandos. - Generar cuádruplo de READ.



WRITE	
#	**Accion semantica**
1	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
2	- Añadir constante de tipo string a la tabla de variables constantes. - Generar dirección constante y hacer push() de ella a la pila de operandos.
3	- Hacer pop() de la pila de operandos. - Generar cuádruplo WRITE con la dirección del operando.

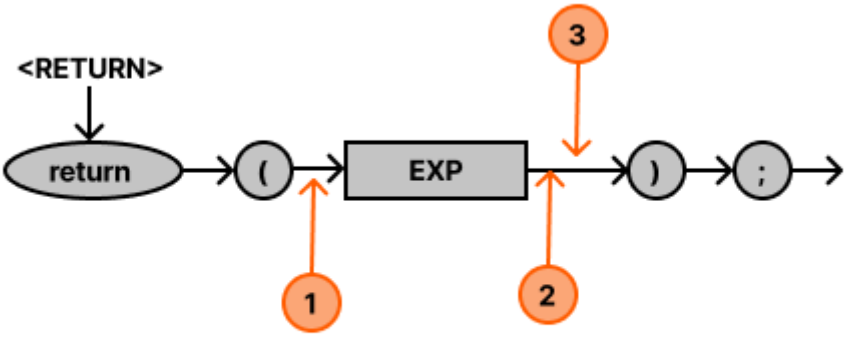


LOOP_COND	
#	**Accion semantica**
1	Hacer push() del número del cuádruple actual a la pila de saltos.
2	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
3	- Hacer pop() de la pila de operandos. - Generar cuádruplo de GOTOF.
4	- Hacer pop() de la pila de saltos. - Agregar número de cuádruplo actual al GOTOF del while. - Hacer pop() de la pila de saltos. - Generar cuádruplo GOTO hacia la expresión antes del GOTOF del while.

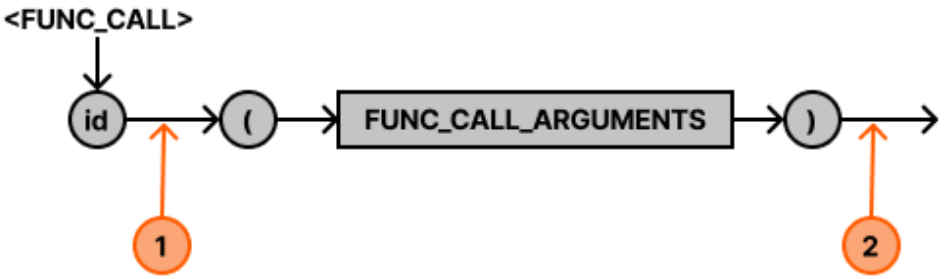


LOOP_RANGE	
#	**Accion semantica**
1	Hacer push() del símbolo “=” a la pila de operadores.
2	- Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores. - Hacer push() de la variable de control a la pila de operandos.
3	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de

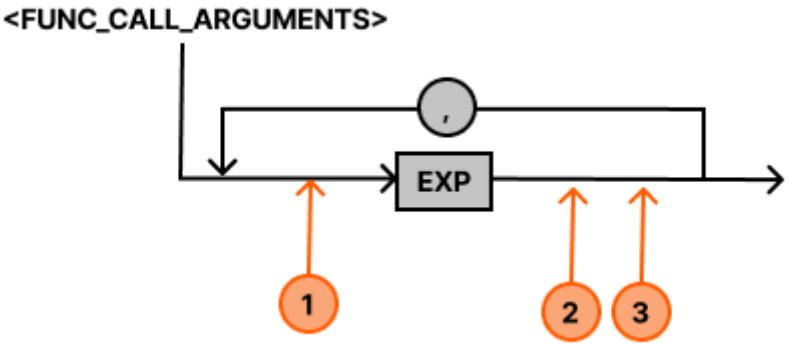
	las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
4	- Hacer pop() de la pila de operandos para obtener el valor final, VF. - Hacer pop() de la pila de operandos para obtener la variable de control, VC. - Generar cuádruplo de comparación entre VC y VF - Hacer push() de VC a la pila de operandos. - Hacer push() del cuádruplo actual a la pila de saltos. - Generar cuádruplo GOTOF. - Hacer push() del cuádruplo actual a la pila de saltos.
5	- Hacer pop() de VC de la pila de operandos. - Generar cuádruplo de suma entre 1 y VC. - Añadir 1 a la tabla de constantes en caso de que no encuentre ya ahí. - Generar cuádruplo de asignación entre VC y la temporal resultante del cuádruplo de suma. - Hacer pop() de la pila de saltos. - Agregar el número de cuádruplo actual al GOTOF del for. - Hacer pop() de la pila de saltos. - Generar cuádruplo GOTO al cuádruplo de comparación del for.



RETURN	
#	**Accion semantica**
1	Hacer push() de ‘stop’ a la pila de operadores.
2	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
3	- Hacer pop() de la pila de operandos. - Obtener dirección de retorno global la función actual. - Generar cuádruplo RET con operando y dirección de retorno.

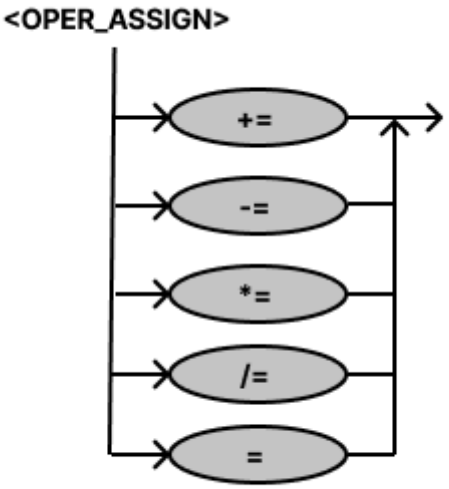


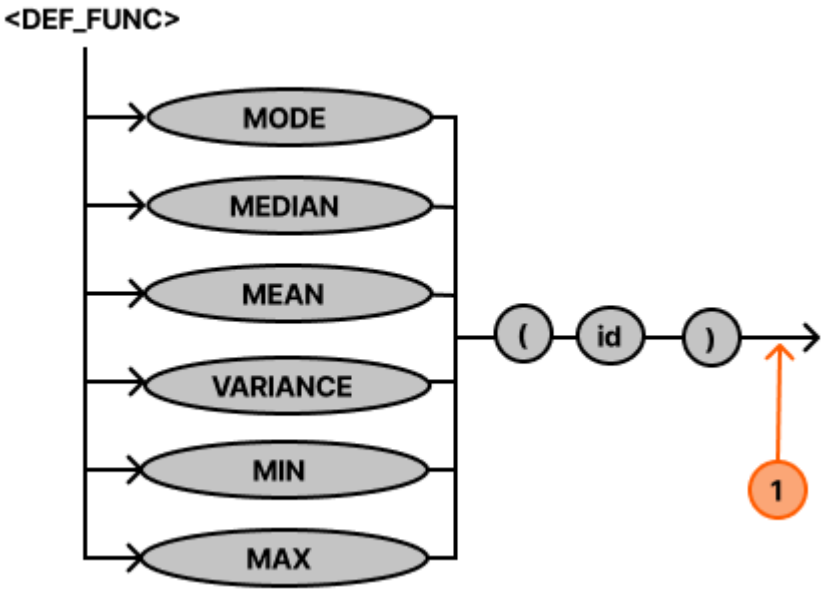
FUNC_CALL	
#	**Accion semantica**
1	- Generar cuádruplo ERA. - Igualar argk = 0.
2	- Comparar argk con tamaño del vector de parámetros de la función. - Generar cuádruplo GOSUB. - Generar cuádruplo de asignación de valor en caso de que la función no sea de tipo void.



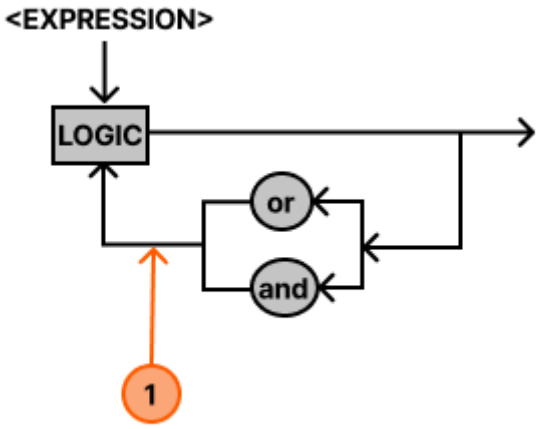
FUNC_CALL_ARGUMENTS	
#	**Accion semantica**
1	Hacer push() de ‘stop’ a la pila de operadores.
2	Hacer pop() de la pila de operadores y generar cuadruplos luego hacer push() de las variables temporales en la pila de operandos hasta encontrar un ‘stop’ o vaciar la pila de operadores.
3	- Hacer pop() de la dirección del argumento de la pila de operandos.

	- Incrementarle 1 a argk. - Generar cuádruplo PARAM con dirección de argumento y argk.

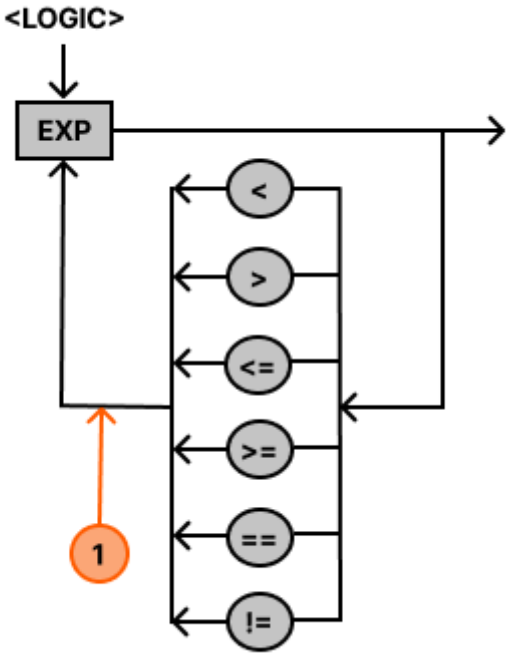




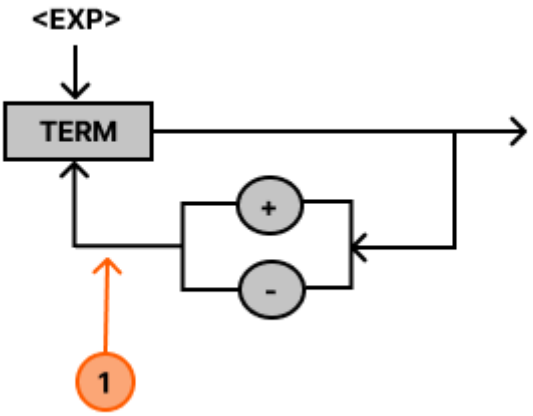
DEF_FUNC	
#	**Accion semantica**
1	- Obtener dimensión del arreglo de la tabla de variables - Generar cuádruplo de función, con el id del arreglo y su dimensión.



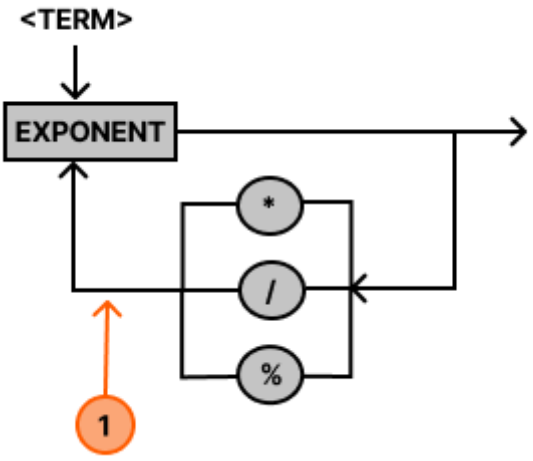
EXPRESSION	
#	**Accion semantica**
1	Hacer push() de operador “or” o “and” a la pila de operadores.



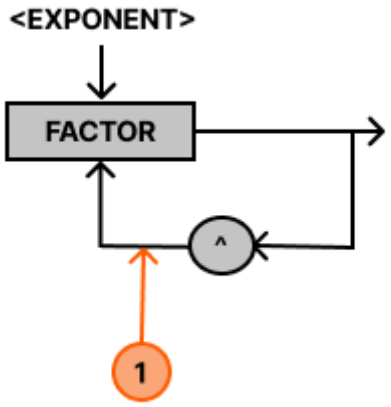
LOGIC	
#	**Accion semantica**
1	Hacer push() de operador “<”, “>”, “<=”, “>=”, “!=” o “==”a la pila de operadores.



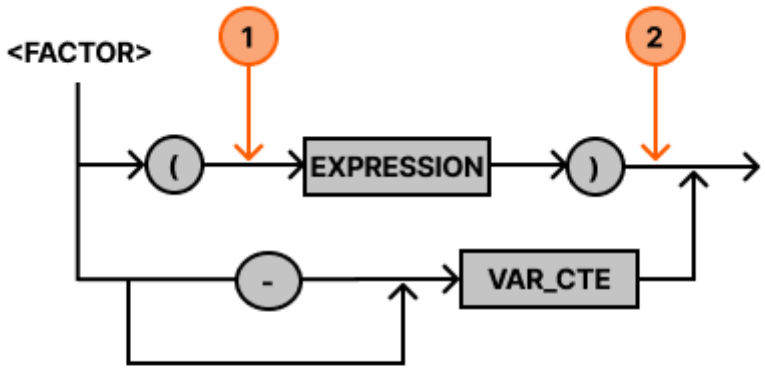
EXP	
#	**Accion semantica**
1	Hacer push() de operador “+” o “-” a la pila de operadores.



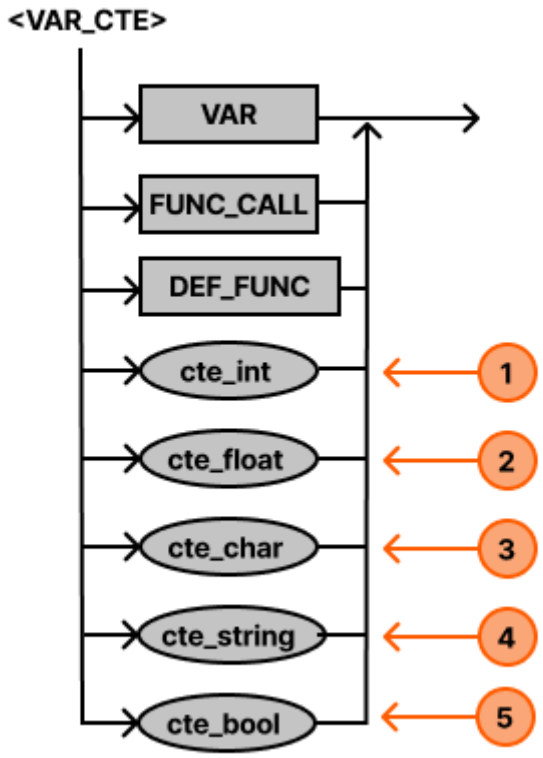
TERM	
#	**Accion semantica**
1	Hacer push() de operador “*”,”/” o “%” a la pila de operadores.



EXPONENT	
#	**Accion semantica**
1	Hacer push() de operador “^” a la pila de operadores.



FACTOR	
#	**Accion semantica**
1	Hacer push() de operador “(“ a la pila de operadores.
2	Hacer pop() de la pila de operadores hasta encontrar un “(“, generar cuadruplos en el proceso e ir metiendo los temporales generados en la pila de operandos.



VAR_CTE	
#	**Accion semantica**
1	- Añadir constante entera a la tabla de constantes - Hacer push() de la constante entera a la pila de operadores.
2	- Añadir constante flotante a la tabla de constantes - Hacer push() de la constante flotante a la pila de operadores.
3	- Añadir constante carácter a la tabla de constantes - Hacer push() de la constante carácter a la pila de operadores.
4	- Añadir constante string a la tabla de constantes - Hacer push() de la constante string a la pila de operadores.
5	- Añadir constante booleana a la tabla de constantes - Hacer push() de la constante booleana a la pila de operadores.

Tablas de consideraciones semánticas

(<,>,<=,>=)	Operador Derecho					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	BOOL	BOOL	error	error	error
	FLOAT	BOOL	BOOL	error	error	error
	BOOL	error	error	error	error	error
	CHAR	error	error	error	error	error
	STRING	error	error	error	error	error

(and, or)	Operador Derecho					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	error	error	error	error	error
	FLOAT	error	error	error	error	error
	BOOL	error	error	error	error	error
	CHAR	error	error	error	BOOL	error
	STRING	error	error	error	error	error

(!=, ==)	Operador Derecho					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	BOOL	BOOL	BOOL	BOOL	BOOL
	FLOAT	BOOL	BOOL	BOOL	BOOL	BOOL
	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
	CHAR	BOOL	BOOL	BOOL	BOOL	BOOL
	STRING	BOOL	BOOL	BOOL	BOOL	BOOL

(+)	Right Operator					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	INT	FLOAT	error	error	error
	FLOAT	FLOAT	FLOAT	error	error	error
	BOOL	error	error	STRING	error	STRING
	CHAR	error	error	error	error	error
	STRING	error	error	STRING	error	STRING

(-,^,*,%)	Operador Derecho					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	INT	FLOAT	error	error	error
	FLOAT	FLOAT	FLOAT	error	error	error
	BOOL	error	error	error	error	error
	CHAR	error	error	error	error	error
	STRING	error	error	error	error	error

(/)	Operador Derecho					
Operador Izquierdo		**INT**	**FLOAT**	**BOOL**	**CHAR**	**STRING**
	INT	FLOAT	FLOAT	error	error	error
	FLOAT	FLOAT	FLOAT	error	error	error
	BOOL	error	error	error	error	error
	CHAR	error	error	error	error	error
	STRING	error	error	error	error	error

3.5. Administración de Memoria en Compilación

Directorio de Procedimientos

El directorio de procedimientos consiste en un diccionario de python (hashtable) este cuenta con nombre, tipo de datos, cuádruplo donde inició (en caso de ser una función definida por el usuario) número de parámetros (en caso de ser funcio), número de variables locales y temporales, un vector de parámetros que es una lista de python, tamaño que consiste en un conteo de locales y temporales de tipos los 5 tipos de datos enteras, flotantes, booleanos. También se guardan las tablas de variables dentro su respectivo procedimiento.Al final del proceso de compilación se agrega este directorio a un json que sirve como el código objeto, que luego es utilizado por la máquina virtual en ejecución.

nombre	Tipo de dato	Quad start	param	local	temp	Vector de parametros	Tam-año	Var table
program	void		0	0	0	[]	...	...
aux_fibonacci	void	1	4	8	3	[‘int’,’int’,’int’,’int’]	...	...
fibonacci	void	18	1	9	5	[‘int’]	...	...
main	void		0	0	0	[]	...	...

tamaño									
li	lf	lb	lc	ls	ti	tf	tb	tc	ts
0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	4	0	2	0	0
4	0	0	0	0	2	0	8	0	0
0	0	0	0	0	0	0	0	0	0

Tabla de Variables

La tabla de variables es un diccionario de python (hashtable) que se encuentra dentro de cada elemento del directorio de procedimientos. Esta tabla consiste en el nombre de la variable, su tipo de dato, su scope dentro del programa, su dirección, y dimensión en caso de ser una arreglo. En ejecución es utilizada por la máquina virtual para acceder a información de las variables.

nombre	Tipo de dato	Scope	direccion	dimension
limit	int	global	1000	0

array	int	local	6000	8
aux	int	local	6008	0

Cuadruplos

Los cuádruplos consisten en una lista de python que guarda tuplas que siempre consisten de 4 elementos. Durante la etapa final de compilación estos cuádruplos se convierten en un dicionario de dicionarios que se guarda en un archivo .json que sirve como nuestro código objeto que luego es usado por la máquina virtual en ejecución.

Durante el proceso de compilación
[(oper, left, right, res), (oper, left, right, res)]

Al momento de guardarse en el código objeto (.json)
{“1”:{"oper": "GOTO","left": null,"right": null,"res": 2},
”2”:{ "oper": "ENDFUNC","left": null,"right": null,"res": null}}

Donde:

- oper: operador
- left: operando izquierdo
- right: operando derecho
- res: resultado

#	operator	Operando izquierdo	Operando derecho	resultado
1	GOTO	Nulo	Nulo	# cuadruplo
2	+	direccion	direccion	Direccion temporal
3	RET	direccion	Nulo	Direccion global
...				

Tabla de Constantes

La tabla de constantes consiste en un diccionario de python (hashtable) con cada tipo de dato del lenguaje (int, float, char, string, bool), para cada tipo de dato hay otro diccionario con la constante como llave y su dirección como valor. Al final del proceso de compilación se agrega esta tabla a un json que sirve como el código objeto, que luego es utilizado por la máquina virtual en ejecución.

Tipo de dato	constante	Dirección de memoria

int	0	16000
	1	16001
float	1.1	17000
	3.14	17001
char	A	18000
	B	18001
string	hola	19000
bool	true	20000

Pila de Operandos

La pila de operandos consiste en una lista de python a la cuales solo se le aplica únicamente las operaciones de append/push y pop, usandola como estructura LiFo (Last in First Out). Se utiliza durante el proceso de compilación para guardar y retirar direcciones de memoria para la generación de cuádruplos.

7002
16000
6000

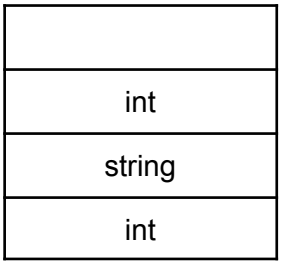
Pila de Operadores

La pila de Operadores consiste en una lista de python a la cuales solo se le aplica únicamente las operaciones de append/push y pop, usandola como estructura LiFo (Last in First Out). Se utiliza durante el proceso de compilación para guardar y retirar operadores para la generación de cuádruplos.

+
(

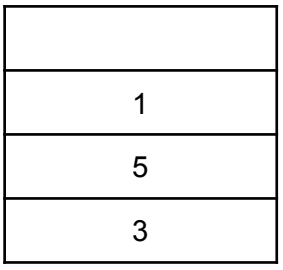
Pila de Tipos de Datos

La pila de tipos de datos consiste en una lista de python a la cuales solo se le aplica únicamente las operaciones de append/push y pop, usandola como estructura LiFo (Last in First Out). Se utiliza durante el proceso de declaración de variables para guardar su tipo de dato en la tabla de variables para después asignarle su dirección de memoria basado en su tipo de dato y scope.



Pila de Saltos

La pila de saltos consiste en una lista de python a la cuales solo se le aplica únicamente las operaciones de append/push y pop, usandola como estructura LiFo (Last in First Out). Se utiliza durante el proceso de compilación para guardar y retirar números de cuadruplos, esto por motivos de control de flujo como bucles(while, for), decisiones (if,else) y llamadas a funciones.



4. DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

4.1. Descripción técnica

Sistema Operativo: Manjaro Linux x86_64
Laptop: HP Pavilion Laptop 15-cw1xxx
Kernel: 5.10.70-1-MANJARO
Shell: zsh 5.8
CPU: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx (8) @ 2.300GHz
GPU: AMD ATI 03:00.0 Picasso
Memoria: 7204MiB / 13964MiB
Lenguaje: Python 3.9.7
Utilerias especiales:

- json
- scipy
- matplotlib
- statistics

4.2. Arquitectura

Tabla de constantes

Al momento que empieza la ejecución de la máquina virtual se guarda la tabla de constantes del código objeto en un diccionario de python donde se realizan todas las llamadas a memoria constante durante la ejecución. Al momento de guardar en el diccionario de python se cambian de lugar las llaves y los valores de la tabla de constantes, ahora siendo la llave la dirección y la constante el valor.

Codigo objeto

tipo	0	16000
	1	16001
float	1.1	17000
	3.14	17001
...		

Maquina Virtual

16000	0
16001	1
17000	1.1
17001	3.14
...	

Directorio de procedimientos

Es la misma estructura que se generó en compilación se utiliza principalmente para obtener los cuádruplos donde inician las funciones, scopes y los tipos de datos de variables y funciones.

nombre	Tipo de dato	Quad start	param	local	temp	Vector de parametros	Tam-año	Var table
program	void		0	0	0	[]	...	...
aux_fibonacci	void	1	4	8	3	[‘int’,’int’,’int’,’int’]	...	...
fibonacci	void	18	1	9	5	[‘int’]	...	...
main	void		0	0	0	[]	...	...

tamaño									
li	lf	lb	lc	ls	ti	tf	tb	tc	ts
0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	4	0	2	0	0
4	0	0	0	0	2	0	8	0	0
0	0	0	0	0	0	0	0	0	0

Diccionario de memoria global

Estructura de datos que almacena la memoria global durante la ejecución del programa. Consiste en un diccionario de python, las direcciones a utilizar se declaran antes de ejecutar los cuádruplos, se les asigna el valor de Nulo. Durante la ejecución del programa su valor cambia al momento que se le asigna un nuevo valor.

direccion	valor
1000	21
3000	False

Lista de diccionario de memoria local

Consiste en una lista de python que se inicializa con un solo diccionario vacío dentro donde se van guardando las variables locales. Cuando se encuentra un cuádruplo ERA se le hace push() a un nuevo diccionario vacío a la lista, sin embargo se siguen accediendo los valores del diccionario previo hasta después recorrer los cuádruplos de PARAM. Cuando se encuentra con un cuádruplo de RET o ENDFUNC se le hace pop() a esta lista de diccionarios, cambiandonos de contexto al procedimiento del cual se llamó a la función.

Indice de lista	Diccionarios en la lista	
1	apuntador	direccion
	6000	31
	8000	False
2	apuntador	direccion
	6000	31
	8000	False

Lista de diccionario de memoria temporal

Consiste en una lista de python que se inicializa con un solo diccionario vacío dentro donde se van guardando las variables temporales. Cuando se encuentra un cuádruplo ERA se le hace push() a un nuevo diccionario vacío a la lista, sin embargo se siguen accediendo los valores del diccionario previo hasta después recorrer los cuádruplos de PARAM. Cuando se encuentra con un cuádruplo de RET o ENDFUNC se le hace pop() a esta lista de diccionarios, cambiandonos de contexto al procedimiento del cual se llamó a la función.

Indice de lista	Diccionarios en la lista	
1	apuntador	direccion
	6000	31
	8000	False
2	apuntador	direccion
	6000	31
	8000	False

Diccionario de apuntadores

Este es un diccionario de python (hashtable) en el cual se guardan los apuntadores. Dentro del ciclo de ejecución si se detecta una variable de tipo apuntador, esta cambia su dirección a la que estaba apuntando el apuntador.

apuntador	direccion
21000	6000
21001	8000

Cuadruplos

Se utiliza la misma estructura generada en el proceso de compilación para recorrer los cuádruplos. La manera en la que se recorren con él instruction pointer ya que cada cuádruplo es parte de un diccionario en el cual un identificador numérico inicializado en 0 actúa como la llave.

Al momento de guardarse en el código objeto (.json)
{“1”:{"oper": "GOTO","left": null,"right": null,"res": 2},
”2”:{ "oper": "ENDFUNC","left": null,"right": null,"res": null}}

Donde:

- oper: operador
- left: operando izquierdo
- right: operando derecho
- res: resultado

#	operator	Operando izquierdo	Operando derecho	resultado
1	GOTO	Nulo	Nulo	# cuadruplo
2	+	direccion	direccion	Direccion temporal
3	RET	direccion	Nulo	Direccion global
...				

Pila de GOSUB

La pila de GOSUB consiste en una lista de python a la cuales solo se le aplica únicamente las operaciones de append/push y pop, usandola como estructura LiFo (Last in First Out).

Se utiliza durante el proceso de compilación para guardar y retirar números de cuadruplos, esto por motivos de control de flujo como bucles(while, for), decisiones (if,else) y llamadas a funciones.

1

5
3

5. PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

PRUEBA #1: Fibonacci

Codigo

```
You, 2 days ago | 1 author (You)
program MyRlike;

function void aux_fibonacci(int: limit,int: curr,int:prev,int:cont){
    vars{
        int: aux;
    }
    if(cont < limit){
        write(curr,",");
        cont += 1;
        aux = prev;
        prev = curr;
        curr += aux;
        aux_fibonacci(limit,curr,prev,cont);
    }

}

function void fibonacci(int: limit){
    vars{
        int: prev,curr,aux;
    }
    if(limit<0){
        write("NEGATIVE NUMBER");
    }
    prev = 1;       You, 2 days ago • Avance Final
    curr = 1;
    if(limit > 1){
       write(prev,",");
    }
    if(limit > 2){
        write(curr,",");
    }

    if(limit > 3){
        aux = prev;
        prev = curr;
        curr += aux;
        aux_fibonacci(limit,curr,prev,2);
    }
}


main(){
   fibonacci(10);
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_fibonacci.alebrije
1,1,2,3,5,8,13,21,34,55,
[gera@garzafox alebrije]$
```

PRUEBA #2: Busqueda Binaria

Codigo

```
program MyRlike;
vars{ int: array[15]; }

function void set_array(){
    vars{ int: i; }
    array[0] = 0;
    for i=1 to len(array) - 1{ array[i] = 2^(i - 1);}
}

function int aux_binary_search(int: target, int: low, int: high){
    vars{ int: mid; }
    if(low <= high){
        mid = low + (high - low) / 2;
        if(array[mid] == target){ return(mid); }
        if(array[mid] > target){ return(aux_binary_search(target,low, mid - 1)); }
        if(array[mid] < target){ return(aux_binary_search(target,mid + 1, high));}
        else{ return(-1);}
    }
    return(-1);
}

function int binary_search(int: target){
    vars{ int: linf,lsup,res;}
    linf = 0;
    lsup = len(array) - 1;
    res = aux_binary_search(target,linf,len(array));
    return(res);
}

main(){
    vars{ int: r,res;}
    set_array();
    read(r);
    res = binary_search(r);
    if(res == -1){
        write("the number isn't in the array","\n");
    }else{
        if(r == array[res]){
           write("success!!!","\n");
           write(r,"==",array[res],"\n");
        }
    }
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_binary_search.alebrije
>128
res 8
success!!!
128==128

[gera@garzafox alebrije]$ python parser.py Test/test_binary_search.alebrije
>3
res -1
the number isn't in the array
```

PRUEBA #3: Bubble Sort

Codigo

```
program MyRlike;
vars{ int: array[20]; }

function void set_array(){
    array[0] = 5;   array[1] = 1; array[2] = 9;   array[3] = -3;
    array[4] = 5;   array[5] = 8; array[6] = 12;  array[7] = -5;
    array[8] = 44;  array[9] = 3; array[10] = 2;  array[11] = 55;
    array[12] = -1; array[13] = 0; array[14] = 18; array[15] = 11;
    array[16] = 7;  array[17] = 100; array[18] = 44; array[19] = 66;
}       You, 2 days ago • Avance Final

function void bubble_sort(){
    vars{ int: prev,curr,aux,i,j,lim; }
    i=1;
    while(i<=len(array)){
        j = 0;
        while(j < len(array) - i){
            if(array[j]>array[j+1]){
                aux = array[j];
                array[j] = array[j+1];
                array[j+1] = aux;
            }
            j+=1;
        }
        i += 1;
    }
}

function void print_array(){
    vars{ int: i; }
    i = 0;
    while(i<len(array)){
        write(array[i],",");
        i += 1;
    }
    write("\n");
}

main(){
   set_array();
   print_array();
   bubble_sort();
   print_array();
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_bubble_sort.alebrije
5,1,9,-3,5,8,12,-5,44,3,2,55,-1,0,18,11,7,100,44,66,
-5,-3,-1,0,1,2,3,5,5,7,8,9,11,12,18,44,44,55,66,100,
```

PRUEBA #4: Quick Sort

Codigo

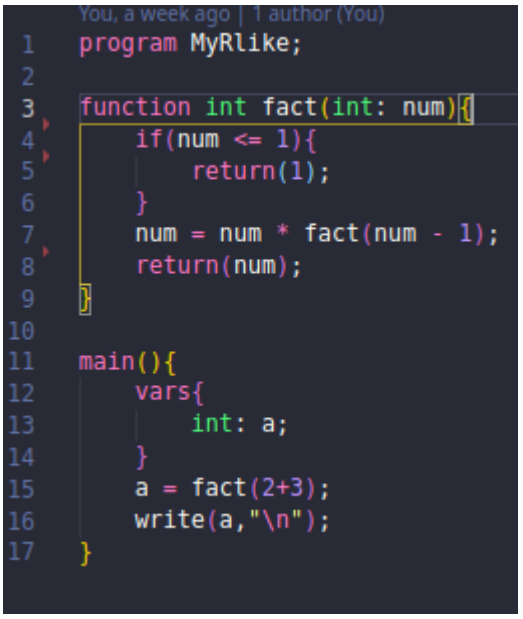
```
program MyRlike;
vars{ int: array[20];}

function void set_array(){
    array[0] = 5;   array[1] = 1; array[2] = 9;   array[3] = -3;
    array[4] = 5;   array[5] = 8; array[6] = 12;  array[7] = -5;
    array[8] = 44;  array[9] = 3; array[10] = 2;  array[11] = 55;
    array[12] = -1; array[13] = 0; array[14] = 18; array[15] = 11;
    array[16] = 7;  array[17] = 100; array[18] = 44; array[19] = 66;
}

function int partition(int: low, int: high){
    vars{ int: pivot, aux,j,i; }
    pivot = array[high];
    i = (low - 1);
    for j=low to high{
        if(array[j] < pivot){
            i += 1;
            aux = array[i];
            array[i] = array[j];
            array[j] = aux;
        }
    }
    aux = array[i+1];
    array[i+1] = array[high];
    array[high] = aux;
    return(high);
}

function void quick_sort(int: start, int: end){
    vars{ int: p; }
    if(start < end){
        p = partition(start,end);
        quick_sort(start,p - 1);
        quick_sort(p + 1, end);
    }
}

function void print_array(){
    vars{ int: i; }
    for i = 0 to len(array){
        write(array[i],",");
    }
    write("\n");
}

main(){
   set_array();
   print_array();
   quick_sort(0,len(array) - 1);
   print_array();
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_quicksort.alebrije
5,1,9,-3,5,8,12,-5,44,3,2,55,-1,0,18,11,7,100,44,66,
-5,-3,-1,0,1,2,3,5,5,7,8,9,11,18,12,44,44,55,66,100,
```

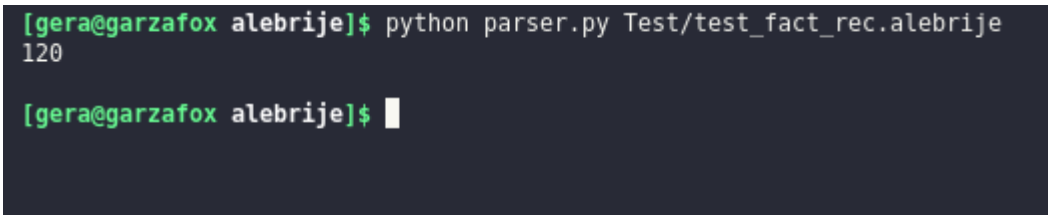
PRUEBA #5: Factorial recursivo

Codigo



```
You, a week ago | 1 author (You)
program MyRlike;

function int fact(int: num){
    if(num <= 1){
        return(1);
    }
    num = num * fact(num - 1);
    return(num);
}

main(){
    vars{
        int: a;
    }
    a = fact(2+3);
    write(a,"\n");
}
```

Resultado



```
[gera@garzafox alebrije]$ python parser.py Test/test_fact_rec.alebrije
120

[gera@garzafox alebrije]$
```

PRUEBA #6: Factorial Iterativo

Codigo

```
program MyRlike;
vars{
    int: w;
}

function int fact(int: num){
    vars{
        int: accum;
    }
    accum = 1;

    while(num > 1){
        accum = accum * num;
        num = num - 1;
    }
    //write('accum: ',accum,'\n');
    return(accum);
}

main(){
    vars{
        int: a,b,c;
    }
    a = fact(4);
    b = fact(3);
    write("a:",a,"\n");
    write("b:",b,"\n");


    // (2 + 24 * 2) * 2 = 100
    write('res: ',(2 + fact(4) * fact(2)) * 2,'\n');

    w = 6;
    w = 2;
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_fact_iter.alebrije
a:24
b:6
res: 100

[gera@garzafox alebrije]$
```

PRUEBA #7: Números Primos

Codigo

```
program MyRlike;
vars{ int: primes[100]; }

main(){
    vars{
        int: limit, list_size, i,j;
        bool: add;
    }

    limit = 100;
    list_size = 1;
    primes[0] = 2;
    for i = 3 to limit{
        add = true;
        for j = 0 to list_size{
            if (i % primes[j] == 0){
                add = false;
            }
        }
        if(add){
            primes[list_size] = i;
            list_size += 1;
        }
    }
    for i = 0 to list_size {
        write(primes[i],",");
    }
}

```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_prime.alebrije
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,
[gera@garzafox alebrije]$
```

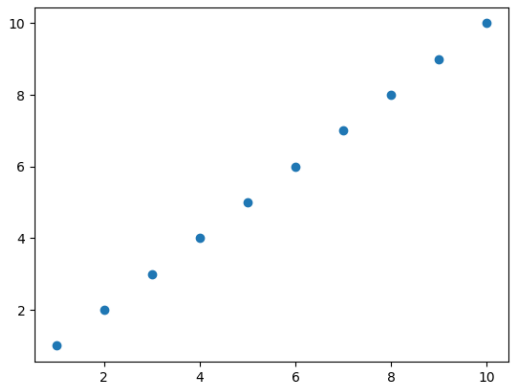
PRUEBA #8: Estadistica

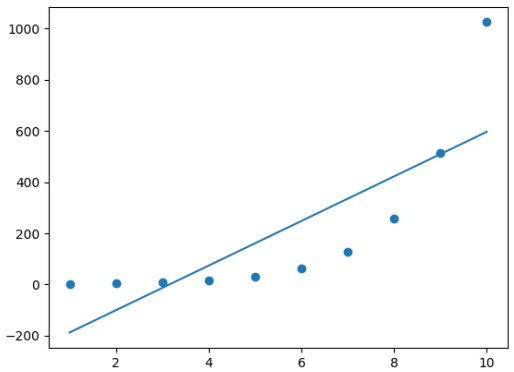
Codigo

```
You, a minute ago | 1 author (You)
program MyRlike;

main(){
    vars {
        int: i,arr[10],x[10],exp[10];
        bool: D,E;
        float: G;
    }
    i = 0;
    while(i<10){
        x[i] = i+1;
        exp[i] = 2^(i+1);
         if (i != 5){
            arr[i] = (i+1)*2;
        }
        i+=1;
    }
    arr[5] = 20;
    write("max: ",max(arr),"\n");
    write("min: ",min(arr),"\n");
    write("mean: ",mean(arr),"\n");
    write("mode: ",mode(arr),"\n");
    write("median: ",median(arr),"\n");
    write("varaince: ",variance(arr),"\n");

    plotxy(x,x);
    regression(x,exp);
}

```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_math.alebrije
max: 20
min: 2
mean: 11
mode: 20
median: 12
varaince: 44
Icon theme "ubuntu-mono-dark" not found.
Icon theme "Mint-X" not found.
Icon theme "elementary" not found.
```





PRUEBA #9: Suma de arreglos
Codigo

```
program MyRlike;

vars{
    int: A[20],B[20],C[20];
}

function void sum_arrays(){
    vars{
        int: i;
    }

    for i=0 to len(C) - 1{
        C[i] = A[i] + B[i];
        write(C[i],",");
    }
    write("\n");
}

main(){
    vars{
        int: i;
    }
    for i=0 to len(A) - 1{
        A[i] = i;
        B[i] = i*2;
    }

    sum_arrays();

}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_sum_arrays.alebrije
0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,

[gera@garzafox alebrije]$
```

PRUEBA #10:Regla trapezoidal
Codigo

```
program MyRlike;

function float f(float: x){
    //0.7853 with limit lower= 0 and upper = 1
    return( 1/(1 + x^2));
}

function float trapezoid(float: x0, float: xn, int: n){
  vars{
      int: i,k;
      float: h, integration;
  }

  h = (xn - x0) / n;

  integration = f(x0) + f(xn);

  for i = 1 to n - 1{
      k = x0 + i*h;
      integration = integration + 2 * f(k);
  }
  integration = integration * h/2;
  return(integration);
}

main(){
    vars{
        int: sub_interval;
        float: llim,ulim,res;
    }
    read(llim);
    read(ulim);
    read(sub_interval);

    res = trapezoid(llim, ulim, sub_interval);
    write(res);
}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_trapezoid.alebrije
>0
>1
>6
0.7916666666666666
[gera@garzafox alebrije]$
```

6. DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO
7. MANUAL DE USUARIO