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 como Alebrije que está basado en R. Este está orientado a jóvenes que buscan aprender los fundamentos de la programación y la capacidad de realizar análisis estadístico básico. El alcance del lenguaje Alebrije debe de incluir: el manejo de expresiones aritméticas, lógicas y/o relacionales; estatutos de interacción de entrada y salida; estatutos de flujo con decisiones y/o ciclos; elementos de cambio de contexto(funciones parametrizables); manejo de elementos no atómicos(arreglos) y funciones predefinidas de estadística para el cálculo de la media, moda, varianza, regresión simple y graficación de puntos en un plano cartesiano.

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 las siguientes expresiones en su orden aritméticas(con PEMDAS) > lógicas > relacionales: - 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 antes de ser utilizadas dentro de una expresión. - Las variables deben de tener un valor asignado previo a su uso.		
Post-Condiciones:	- Los operandos deben de tener tipos de datos compatibles con entre sí 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.		
Pre-Condiciones:	- La variable debe de estar previamente definida antes de la asignación. - La expresión que se le asigna debe de ser válida. - El resultado de la expresión debe de ser aceptado por el tipo de dato de la variable.		
Post-Condiciones:	La variable debe de recordar su último valor asignado en caso de ser llamada durante otra expresión.		

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 una o más variables.		
Pre-Condiciones:	- La o las variables a leer deben de estar previamente definidas. - El tipo de dato asignado a la variable debe de ser compatible con el tipo de datos de la variable.		
Post-Condiciones:	- Las variables deben de recordar sus nuevos valores asignados.		

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 y funciones en caso de ser utilizadas 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 de decisión 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 del if o del else. - 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 bucles de rango como el for, el cual es capaz de repetir más estatutos dentro de sí mismo incluyendo mas 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 for.		

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. - Todas las funciones deben de tener un tipo de dato. - Todos los parámetros en la declaración de una función deben de tener algún tipo de dato. - No pueden existir 2 funciones con el mismo nombre.		
Post-Condiciones:	- 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 sin valor de retorno (void).		
Pre-Condiciones:	- La función debe de ser declarada como tipo de dato void, en caso de ser definida por el usuario. - Las mismas que R8		
Post-Condiciones:	- Las funciones Void no pueden tener retornar valores. - Las mismas que R8		

ID	R10	**Prioridad:**	Media
Nombre	Ejecución de funciones parametrizables recursivas	**Creado por:**	Gerardo
Descripción:	- El lenguaje debe de soportar funciones con recursión.		
Pre-Condiciones:	- Las mismas que R8		
Post-Condiciones:	- La función debe de tener alguna manera proporcionada por el usuario de detener el ciclo de recursión. - Las mismas que R8		

ID	R11	**Prioridad:**	Alto
Nombre	Manejo de arreglos	**Creado por:**	Gerardo
Descripción:	El lenguaje debe de soportar el manejo de estructuras de datos no atómicos en el caso de Alebrije arreglos.		
Pre-Condiciones:	- El arreglo debe de estar declarado antes de llamarse. - El arreglo debe de tener una dimensión asociada a él.		
Post-Condiciones:	- Se debe verificar que el índice o el resultado de la expresión 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 distinta.		
Pre-Condiciones:	- Las mimosas que R8		
Post-Condiciones:	- Las mimosas que R8		

ID	R13	**Prioridad:**	Media
Nombre	Funciones Estadísticas(Media, Moda, Varianza, Regresión, 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 y tipo correcto de argumentos a cada función.		
Post-Condiciones:	- Las funciones regression y plotxy deben de ser tratadas como funciones de tipo void.		

TEST CASES

ID	T1	**Prioridad:**	Alta
Nombre	Fibonacci recursion por cola	**Creado por:**	Gerardo
Descripción:	Programa que calcula la secuencia de números fibonacci por medio de una función auxiliar y la recursión por cola. 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		

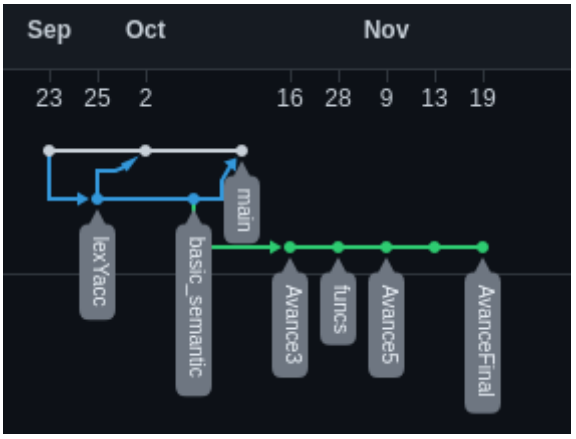
ID	T10	**Prioridad:**	Baja
Nombre	Regla trapezoidal	**Creado por:**	Gerardo

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

ID	T11	**Prioridad:**	Baja
Nombre	Fibonacci doble recursion	**Creado por:**	Gerardo
Descripción:	Este programa imprime en pantalla la secuencia de números fibonacci hasta el número que dio el usuario. Utiliza recursión doble para su operación.		
Requisitos aplicables:	R1,R2,R3,R4,R5, R9, R10, R12		
Entradas de usuario:	Numero entero		

1.3. Descripción del Proceso

Durante el proceso de desarrollo de este proyecto se utilizó la herramienta de control de versiones git y el servicio de repositorios remotos Github. Para el desarrollo local se utilizó el editor de texto VS Code y la terminal de linux para el desarrollo. Se trabajó de manera constante en base al calendario de entregas.



Grafo de commits en repositorio remoto

Git log

```
commit 8c2f36b0da2b82c431d825b3426a6ef753b7982b
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Sun Nov 21 23:28:44 2021 -0600

        Avanze 7
```

```
commit 673779f34aa47408641780c23d328558edadf35e
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Fri Nov 19 23:18:29 2021 -0600

        Avance Final
```

```
commit 12611bb134ad90e0f36606d21b8d13a4acd2bbf9
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Sat Nov 13 21:37:09 2021 -0600

        VM memory model had a 180º redo, array, recursion and subfunction
calls seem to work properly
```

```
commit a534f5317772de9c5b3fda537e12de825dd44344
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Tue Nov 9 21:44:56 2021 -0600

        Avance 5
```

```
commit 3fc44da9c0926174ef81dc4cb3d2000eb1a0c07f
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Thu Oct 28 21:55:34 2021 -0500

        add vm, recursion not working yet
```

```
commit 74a4813a38d54bdb8ac73aaf7c550b8cc269779b
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Sat Oct 16 23:57:53 2021 -0500

        Added Neuralgic Points for conditionals
```

```
commit 3a04e3c676876e4a88b6b5c56b4af1dbdb9aa0ac
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Sat Oct 2 23:50:37 2021 -0500

        Add procedure directory, variable table and semantic cube
```

```
commit cfe06b099eb33bb70fa6b73b751f69cf8eaedcaa
Author: Gerardo Galan Garzafox <gerardogalangarzafox@gmail.com>
Date:   Sat Sep 25 19:11:48 2021 -0500

        Coded lexer and parser
```

```
commit f4b3eabf0b8a8152949faa856e0227ef8b6ac918
Author: Gerardo Galan <43053794+gggfox@users.noreply.github.com>
Date:   Thu Sep 23 11:29:26 2021 -0500

        Initial commit
```

Semana #	Avance
1	Diagramas de sintaxis, gramática, lexer y parser inicial.
2	Puntos Neurálgicos incompletos para expresiones y decisiones.
3	Directorio de procedimientos, tabla de variables, mejora en puntos neurálgicos.
4	Puntos neurálgicos para ciclos completos, puntos neurálgicos para funciones incompletas y definición inicial del mapa de memoria.
5	Máquina virtual inicial, código objeto, ejecución de aritmética, decisiones y ciclos.
6	Cambio en manejo de memoria, ejecución de funciones y arreglos en máquina virtual.
7	Documentación 1era version, funciones predefinidas, pruebas

Compromisos

- Tratar de evitar tener líneas de código mayores a 80 columnas.
- Poner comentarios en las funciones.
- Utilizar una herramienta de control de versiones.
- Tener al menos 10 pruebas de funcionamiento.
- Usar inferencia de tipos de datos en python.
- Apegarse al calendario de proyecto.

Reflexion

Sin duda este ha sido uno de los proyectos más grandes en los que he trabajado de manera individual, aunque se me viene a la mente uno del semestre pasado y al igual que ese proyecto lo importante es la consistencia. Durante la clase y la realización del proyecto regresaron a mi mente varios temas de mi clase de organización computacional, como el parecido de los cuádruplos a un emulador del lenguaje Marie que vimos durante esa clase. En lo personal me sorprendieron varias cosas de la clase que se aplicaron en el proyecto, como la manera en la que las funciones regresan valores con el uso de variables globales, y la manera en la que se usan los GOTO 's para controlar el flujo del programa. Todo esto, creo yo, nos da una muestra de lo que hay realmente detrás de las bambalinas de un lenguaje de programación de alto nivel y lo importante que es entenderlo, porque además de ser muy gratificante, puede además llegar a ser muy útil.

Firma

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
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	LESS	Menos que	<
30	GREATER	Mayor que	>
31	LESS_EQ	Menor o igual que	<=
32	GREATER_EQ	Mayor o igual que	>=
33	EQUIVALENT	Equivalnete a	==
34	DIFFERENT	Differente a	!=
35	EQUAL	igual	=
36	MULT	multiplicacion	*
37	DIV	division	/
38	PLUS	suma	+
39	MINUS	resta	-
40	REMAINDER	resto	%
41	EXP	exponencial	^
42	MULT_EQ	Multiplicar e igualar	*=
43	DIV_EQ	Dividir e igualar	/=
44	PLUS_EQ	Sumar e igualar	+=
45	MINUS_EQ	Restar e igualar	-=
46	L_BRACE	Llave izquierda	{
47	R_BRACE	Llave derecha	}
48	L_BRACKET	Corchete izquierdo	[
49	R_BRACKET	Corchete derecho	]
50	L_PAR	Parentheses izquierdo	(
51	R_PAR	Parenthesis derecho	)
52	COLON	2 puntos	:
53	SEMICOLON	Punto y coma	;
54	COMMA	coma	,

55	ID	identificador	[a-ZA-Z][a-zA-Z_0-9]*
56	CTE_INT	Constantine entera	-?\d+
57	CTE_FLOAT	Constantine flotante	-?\d+\.\d+
58	CTE_BOOL	Constate booleana	(True| False | true | false)
59	CTE_CHAR	Constaten caracter	(\".\" | \'.\')
60	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
→ char
→ bool
→ string

<VAR>
A → id
→ id [EXP]

<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>
A → while (EXPRESSION) BLOCK

<LOOP_RANGE>
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

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

<EXPONENT>
A → FACTOR B
B → ^ A
→epsilon

<FACTOR>
A → (EXPRESSION)
→ 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:.

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

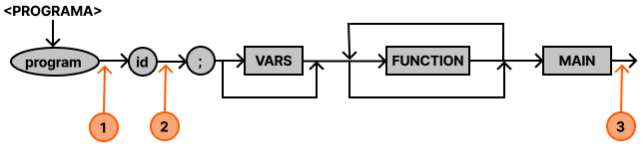
Locales	
int	6000-6999
float	7000-7999
bool	8000-8999
char	9000-9999
string	10000-10999

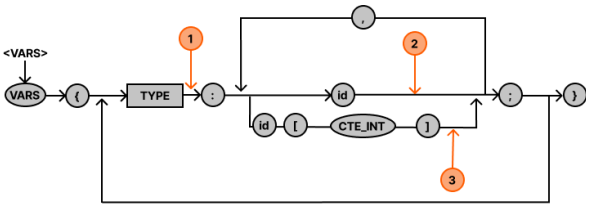
Temporales	
int	11000-11999
float	12000-12999
bool	13000-13999
char	14000-14999
string	15000-15999

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

Apuntadores
21000...

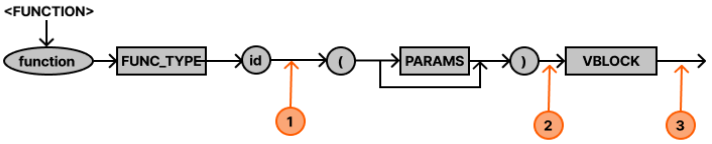
Diagramas de Sintaxis y acciones semánticas

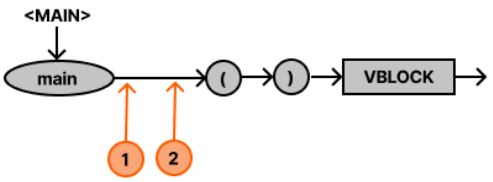




PROGRAM	
#	**Accion semantica**
1	Cambiar procedimiento actual a ‘program’.
2	-saltos.push(cuadruplos.length) -quads.push((“GOTO”,_,_,_))
3	quads.push((“PROGEND”,_,_,_))

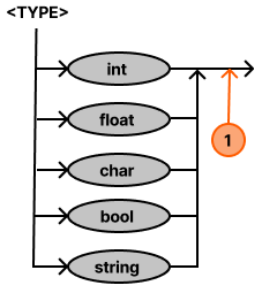
VARS	
#	**Accion semantica**
1	Cambiar el tipo de dato actual.
2	Añadir variable a la tabla de variables de la función actual.
3	Añadir id del arreglo y su dimensión a la tabla de variables de la función actual.

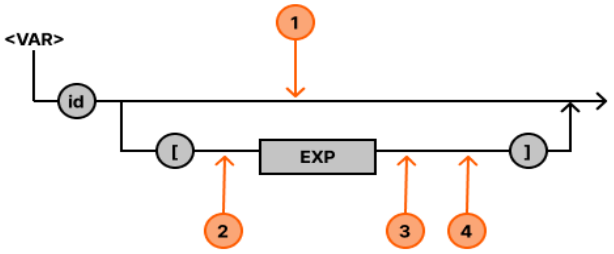




FUNCTION	
#	**Accion semantica**
1	Cambiar procedimiento actual a id de función.
2	Añadir posición del cuádruplo actual de la función al directorio de procedimientos.
3	quads.push((“FUNCEND”,_,_,_))

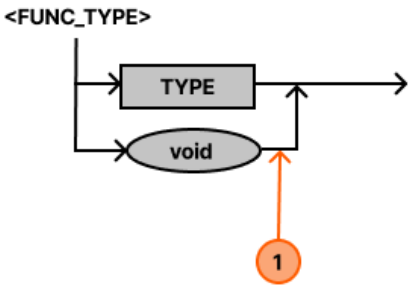
MAIN	
#	**Accion semantica**
1	Cambiar el procedimiento actual a ‘main’.
2	indice=saltos.pop() quads[indice][3]=quads.length

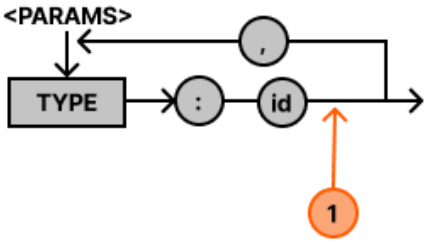




TYPE	
#	**Accion semantica**
1	tipoDatos.push(tipo)

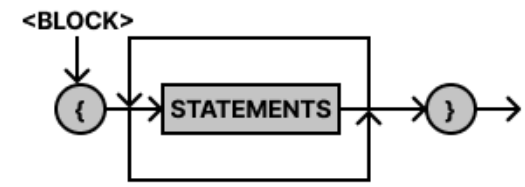
VAR	
#	**Accion semantica**
1	operandos.push(id)
2	operadores.push(‘stop’).
3	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
4	índice = operandos.pop() quads.push((“VER”,índice,linf,lsup) quads.push(“+”,índice,dirB(),ptr)

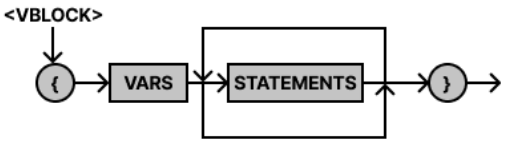


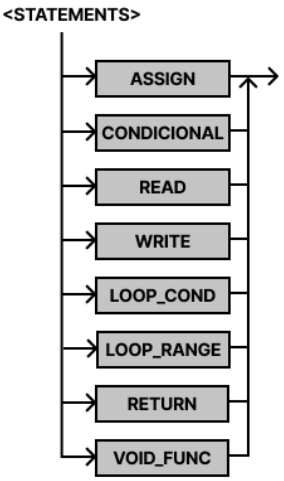


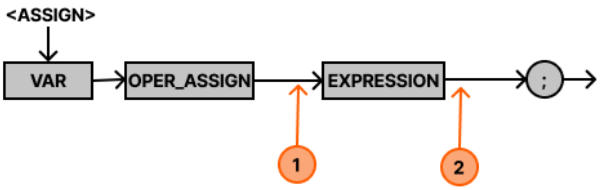
FUNC_TYPE	
#	**Accion semantica**
1	tipoDatos.push(‘void’)

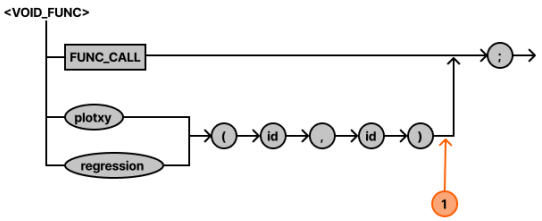
PARAMS	
#	**Accion semantica**
1	-Agrega id y tipo del parámetro a la tabla de variables de la función actual. -Agrega tipo al vector de parámetros de la función.





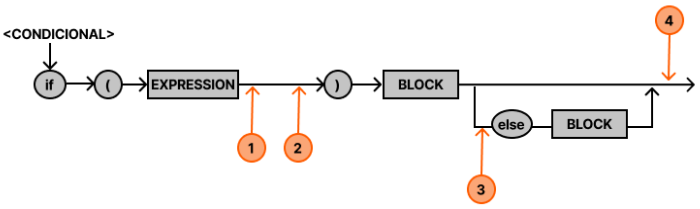


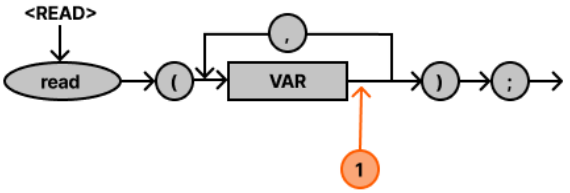




ASSIGN	
#	**Accion semantica**
1	operadores.push(operador)
2	while operadores.length>0 and oper!='stop': generar_cuádruplo(operadores.pop())

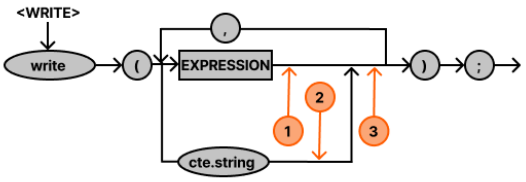
VOID_FUNC	
#	**Accion semantica**
1	Generar cuádruplos de PLOTXY o REGRESSION.

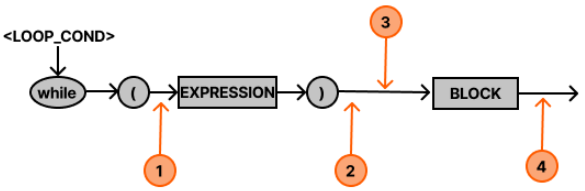




CONDICIONAL	
#	**Accion semantica**
1	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
2	-saltos.push(quads.length) -quads.push((“GOTOF”,operandos.pop(),_,_)
3	-saltos.push(quads.length) -quads[saltos.pop()][3] = quads.length -quads.push((“GOTO”,_,_,quads.length))
4	-quads[saltos.pop()][3] = quads.length

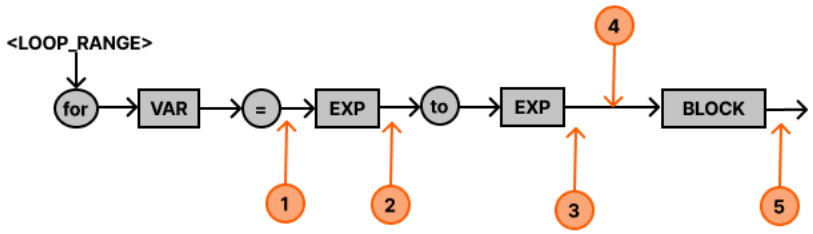
READ	
#	**Accion semantica**
1	-operando = operandos.pop(). -quads.push((“READ”,_,_,operando))



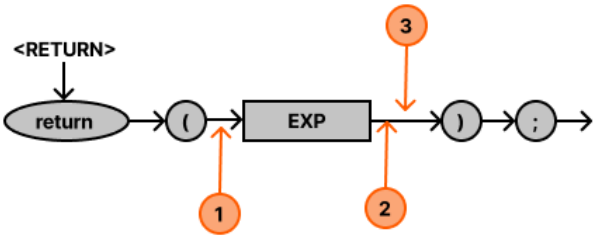


WRITE	
#	**Accion semantica**
1	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
2	-Añadir constante de tipo string a la tabla de variables constantes. -operandos.push(constante)
3	-operando = operandos.pop() -quads.push((“WRITE”,_,_,operando))

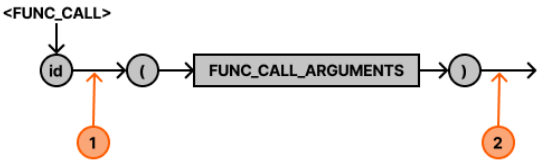
LOOP_COND	
#	**Accion semantica**
1	saltos.push(quads.length)
2	while operadores.length>0 and oper!=‘stop’: generar_cuádruplo(operadores.pop())
3	-cond = operandos.pop() -quads.push((“GOTOF”,cond,_,_))
4	-índice, cond = saltos.pop(), saltos.pop() -quads.push((“GOTO”,_,_,cond)) -quads[índice][3] = quads.length

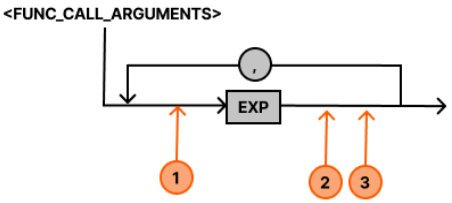


LOOP_RANGE	
#	**Accion semantica**
1	operadores.push(“=”)
2	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
3	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
4	- VF, VC = operandos.pop(),operandos.pop() - quads.push((“<”,VC,VF,Tx)) - operandos.push(CV) - quads.push((“GOTOF”,Tx,_,_)) - saltos.push(quads.length) - saltos.push(quads.length - 1)
5	- VC=operandos.pop() - cuádruplos.push((“+”,VC,1,Tx)) - quads.push((“=”,Tx,_,VC)) - índice,comp =saltos.pop(),saltos.pop() - quads.push((“GOTO”,_,_,comp)) - quads[índice][3] = quads.length



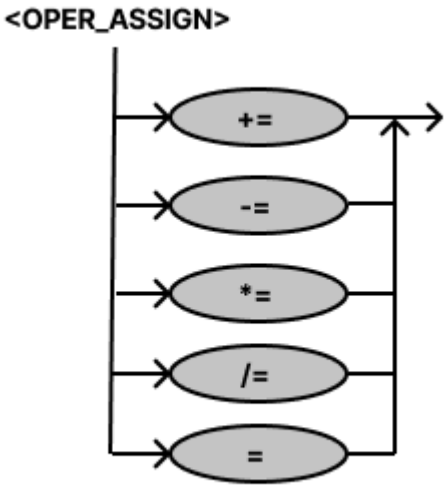
RETURN	
#	**Accion semantica**
1	operadores.push(‘stop’)
2	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
3	-funcDir = procDir.getFuncDir(procDir.funcActual) -quads.push((“RET”,operandos.pop(),_,funcDir)

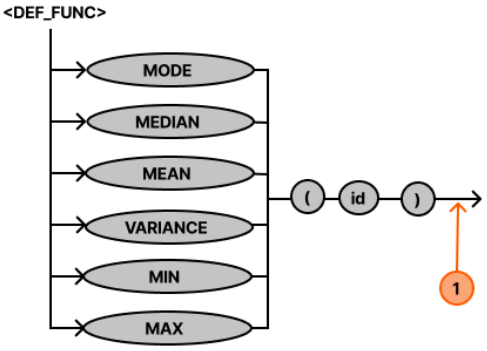




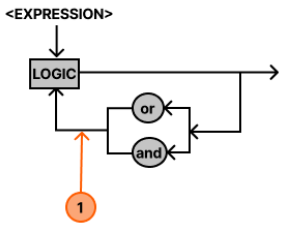
FUNC_CALL	
#	**Accion semantica**
1	-quads.push((“ERA”,_,_,id)) -argk = 0.
2	-si (argk != vectorParam.length) lanzar error -quads.push((“GOSUB”,quads.length,_,id)) -si no es void: quads.push((“=”,procDir.getRet(id),_,Tx)) operandos.push(Tx)

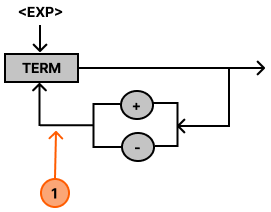
FUNC_CALL_ARGUMENTS	
#	**Accion semantica**
1	operadores.push(‘stop’)
2	while operadores.length > 0 and oper != ‘stop’: generar_cuádruplo(operadores.pop())
3	-arg = operandos.pop() -argk++ -quads.push((“PARAM”,arg,_,argk))

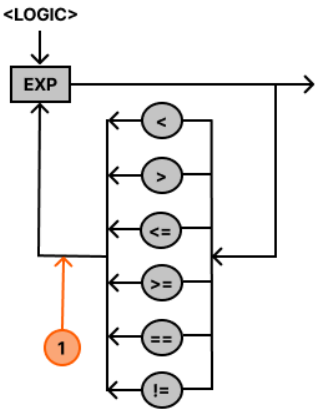


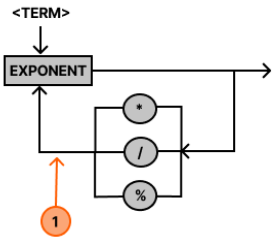


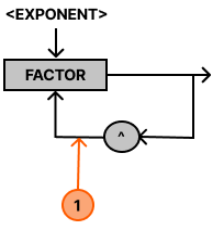
DEF_FUNC	
#	**Accion semantica**
1	-Obtener dimensión del arreglo de la tabla de variables. -Generar cuádruplo de función, con el id del arreglo y su dimensión.



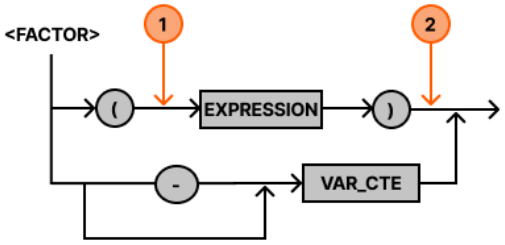


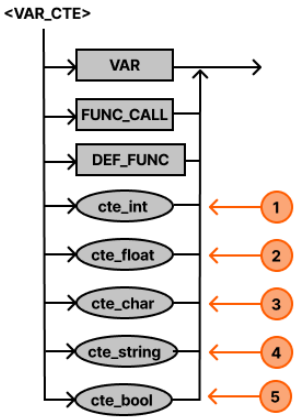






EXPRESSION, LOGIC, EXP, TERM, EXPONENT	
#	**Accion semantica**
1	operadores.push(simbolo)





FACTOR	
#	**Accion semantica**
1	operadores.push(“(“)
2	while operandos.length > 0 and oper != “(“ and priority(oper): generar_cuádruplo(operadores.pop())

VAR_CTE	
#	**Accion semantica**
1, 2, 3, 4, 5	-Añadir a la tabla de constantes según su tipo. -operadores.push(constante)

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) que guarda información sobre procedimientos: nombres, tipo de datos, cuádruplo donde inicia, número de parámetros, número de variables locales, número de variables temporales, un vector de parámetros que es una lista de python. También se guardan las tablas de variables dentro su respectivo procedimiento. Esta estructura se utiliza para múltiples operaciones durante la compilación, una de ellas es verificar el número de argumentos que se usan en la llamada a una función concuerde con el vector de parámetros. Al final del proceso de compilación se genera un código objeto, un .json, donde se incluye este directorio de procedimientos que luego es utilizado por la máquina virtual durante el proceso de ejecución.

nombre	Tipo de dato	Quad start	param	local	temp	Vector de parametros	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	[]	...

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: los nombres de las variables, sus tipos de datos, sus alcances dentro del programa, sus direcciónes y sus dimensiones en caso de que sean arreglos. Cuando se declara una variable se agrega a la tabla de variables, esta es usada durante el proceso de compilación para obtener las direcciones de memoria de las variables. En ejecución es utilizada por la máquina virtual para acceder a información de las variables, como declaración de variables globales en caso de que existan.

nombre	Tipo de dato	Alcanze	direccion	dimension
limit	int	global	1000	0
array	int	local	6000	8
aux	int	local	6008	0

Lista de Cuádruplos

La lista de cuadruplos es una lista de python que va guardando los cuádruplos como tuplas que siempre consisten de 4 elementos, estos siendo el operador, el operando izquierdo, el operando derecho y el resultado. 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 código objeto que luego es usado por la máquina virtual durante el proceso de ejecución para ejecutar archivos Alebrije.

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

#	Operador	Operando izquierdo	Operando derecho	Resultado
1	GOTO	Nulo	Nulo	# cuadruplo
2	+	direccion	direccion	Temporal
3	RET	Temporal	Nulo	Direccion
...	...	...	...	...

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) como llave y como valor otro diccionario(hashtable) con la constante como llave y su dirección como valor. Al final del proceso de compilación se agrega esta tabla al código objeto, el .json, que luego es utilizado por la máquina virtual en ejecución.

Tipo de dato	constante	Dirección de memoria
int	0	16000
	1	16001
	14	16002
float	1.1	17000
	3.14	17001
char	A	18000
	B	18001
string	hola	19000
bool	true	20000

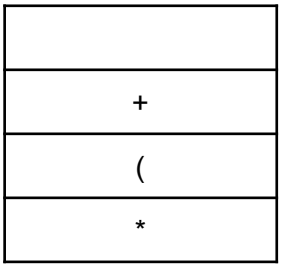
Pila de Operandos

La pila de operandos consiste en una lista de python a la cual únicamente se le aplican las operaciones de append/push y pop, usandola como una 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

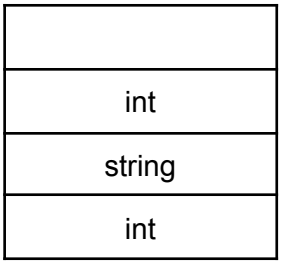
Pila de Operadores

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



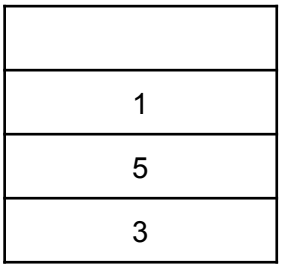
Pila de Tipos de Datos

La pila de tipos de datos consiste en una lista de python a la cual únicamente se le aplican las operaciones de append/push y pop, por lo que se maneja como una 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 cual únicamente se le aplican las operaciones de append/push y pop, usándola como una estructura LiFo (Last in First Out). Se utiliza durante el proceso de compilación para guardar y retirar números de cuadruplos, esto para manejar el control de flujo con elementos 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
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 números de cuádruplos donde inician las funciones, y las tablas de variables de los distintos procedimientos.

nombre	Tipo de dato	Quad start	param	local	temp	Vector de parametros	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	[]	...

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 realiza una asignación.

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 realiza un push() de un nuevo diccionario vacío en la lista, sin embargo se siguen accediendo los valores del diccionario previo hasta después de haber recorrer todos cuádruplos de PARAM antes del cuádruplo del GOSUB. Cuando se encuentra con un cuádruplo con operador RET o ENDFUNC se le hace pop() a esta lista de diccionarios, y a la pila de GOSUB para cambiar de contexto al cuádruplo del cual se llamó a la función.

Indice de lista	Diccionarios en la lista	
1	6000	31
	8000	False
2	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 con el operador ERA se realiza un push() de un nuevo diccionario vacío, sin embargo se siguen accediendo los valores del diccionario previo hasta después de haber recorrido todos los cuádruplos de PARAM antes de llegar al GOSUB. Cuando se encuentra con un cuádruplo con operador RET o ENDFUNC se realiza un pop() de esta lista de diccionarios, también se realiza un pop() de la pila de gosubs para cambiar el contexto al cuádruplo del cual se llamó a la función.

Indice de lista	Diccionarios en la lista	
1	6000	31
	8000	False
2	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 dirección es de tipo apuntador, esta cambia su dirección a la que estaba apuntando el apuntador.

apuntador	direccion
21000	6000
21001	8000

Lista de cuádruplos

Se utiliza la estructura generada en el proceso de compilación, diccionario de diccionarios, para recorrer los cuádruplos. La manera en la que se recorren es con el uso del instruction pointer, el cual se mueve a través de los cuádruplos. Luego se utilizan una serie de condicionales, similar a un switch, para ejecutar las instrucciones del cuádruplos dependiendo de su operador.

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

● 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 cual únicamente se le aplican las operaciones de append/push y pop, por lo que se maneja como una estructura LiFo (Last in First Out). Se utiliza durante el proceso de ejecución para manejar los cambios de contexto. Una vez que se lee un cuádruplo de RET o ENDFUNC, se realiza un pop() de esta pila para mover al instruction pointer al cuádruplo en esa posición.

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 15 - 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 = 15 - 1;
    res = aux_binary_search(target,linf,15);
    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;
}

function void bubble_sort(){
    vars{ int: prev,curr,aux,i,j,lim; }
    i=1;
    while(i<=20){
        j = 0;
        while(j < 20 - 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<20){      You, 2 minutes ago • Uncommitted changes
        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

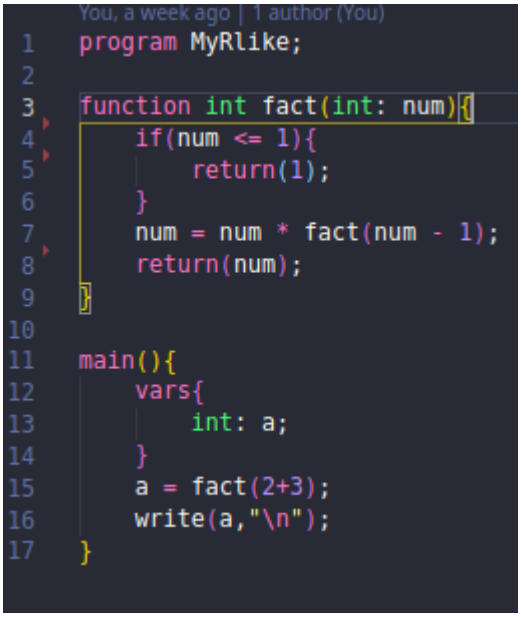
```
You, 4 minutes ago | 1 author (You)
program MyRlike;        You, 5 days ago • Avance Final
vars{ int: array[20];}

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 20{
        write(array[i],",");
    }
    write("\n");
}

main(){
   set_array();
   print_array();
   quick_sort(0,19);
   print_array();
}
```

Resultado

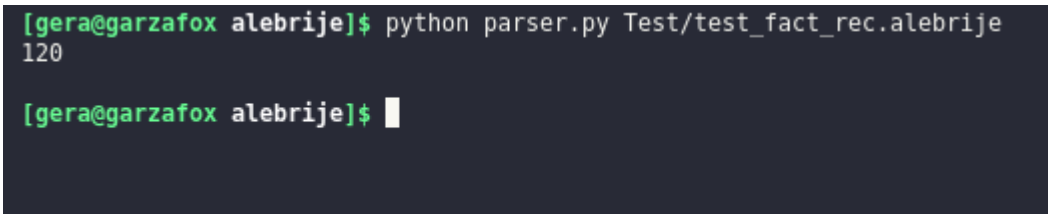
```
[gera@garzafox alebrije]$ python parser.py Test/test_quicksort.alebrije
5,1,9,-3,5,8,12,-5,44,3,2,55,-1,0,18,11,7,100,44,66,
-5,-3,-1,0,1,2,3,5,5,7,8,9,11,18,12,44,44,55,66,100,
```

PRUEBA #5: Factorial recursivo

Codigo



Resultado



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]$
```

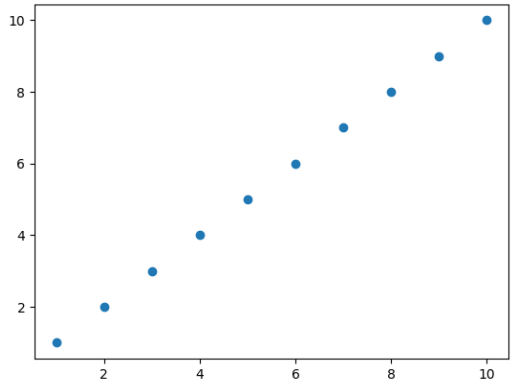
PRUEBA #8: Estadistica

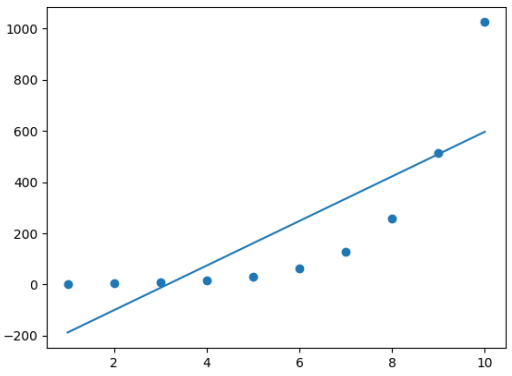
Codigo

```
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 20 - 1{
        C[i] = A[i] + B[i];
        write(C[i],",");
    }
    write("\n");
}

main(){
    vars{
        int: i;
    }
    for i=0 to 20 - 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]$
```

PRUEBA #11: Fibonacci doble recursion

Codigo

```
program MyRlike;

function int fib(int: n){
    vars{
        int: a,b;
    }

    if(n < 2){
        return(n);
    }else{
        return(fib(n - 1) + fib(n - 2));
    }
}

main(){
    vars{
        int: a,b;
    }
    read(b);
    for a = 1 to b{
        write(fib(a), ",");
    }

}
```

Resultado

```
[gera@garzafox alebrije]$ python parser.py Test/test_fibonacci_rec.alebrije
>10
1,1,2,3,5,8,13,21,34,
[gera@garzafox alebrije]$
```

6. DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO

Máquina virtual métodos get_mem(int) y save(int, any)

Funciones save() y get_mem(), estas funciones son cruciales dentro de la máquina virtual, ya que son las que administran la memoria. Antes en una versión previa de memoria existía un arreglo con 21,000 casillas que resultó en complicaciones al momento de intentar manejar los cambios de contexto. Sin embargo con estas 2 funciones realizar cambios se volvió significativamente más simple, la memoria se vuelve un poco más abstracta, pero con el beneficio de ser más fácil de realizar cambios en esta. En save se recibe una dirección de memoria entera y un valor que puede ser de cualquier tipo de dato, esta no regresa ningún valor, únicamente guarda el parámetro valor en el diccionario correcto dependiendo del rango de su dirección de memoria. La función get_mem() recibe únicamente un número entero que es dirección de memoria, con la cual se obtiene el scope y se regresa el valor guardado en el diccionario correcto.

```python
def save(self, addr: int, value) -> None:
    """saves a value in a memory address

    Args:
        addr (int): memory address
        sol ([type]): value
    """
    if type(addr) != int or addr < 1000:
        return addr

    value = self.clean_datatype(addr,value)

    if self.is_glob(addr):
        self.glob[addr] = value
    elif self.is_local(addr):
        self.local[self.era][addr] = value
    elif self.is_temp(addr):
        self.temp[self.era][addr] = value
    elif self.is_pointer(addr):
        self.pointer[addr] = value

def get_mem(self, addr: int):
    """returns the value of a memory address

    Args:
        addr (int): memory address

    Returns:
        [type]: value
    """
    if addr == None or type(addr) != int:
        return addr
    elif self.is_glob(addr):
        return self.glob.get(addr)
    elif self.is_temp(addr):
        return self.temp[self.era].get(addr)
    elif self.is_local(addr):
        # expression is param
        if self.era != len(self.gosub):
            return self.local[self.era - 1].get(addr)
        # default
        else:
            return self.local[self.era].get(addr)
    elif self.is_const(addr):
        return self.const.get(addr)
    elif self.is_pointer(addr):
        return self.pointer.get(addr)
```

Parser función de punto neurálgico np_end

Este punto de neuralgico lo utilizo en 8 de las 30 reglas de mi gramática, se utiliza para definir al fin de una expresión en las reglas. Su modo de operación en simple mientras existan operadores en la pila de operadores hace un pop() de esta, si encuentra un operador de tipo “stop” que actúa como un fondo falso que detiene el ciclo, si el operador es de asignación crea cuádruplo de asignación, en caso de ser asignacion especial crea 2 cuádruplos, para cualquier otro caso llama a la función gen_quad() que es la que se encarga de generar cuadruplos en compilación. Este punto neurálgico no regresa nada solo manipula el stack global de operadores, la lista global de cuádruplos además del objeto global de directorio de procedimientos que se encarga de todo lo que tenga que ver con variables y funciones.

```
def p_np_end(p):
    'np_end : '
    """Generates quadruples until the operator stack is empty or finds a stop
    """
    while len(stack.operators) > 0:
        oper = stack.operators.pop()
        if oper == 'stop':
            break
        if oper == '=':
            gen_assign()
        elif oper in ['+=', '-=', '*=', '/=']:
            (right, left) = stack.get_RL_operands()
            temp = proc_dir.gen_temp(oper[0], right, left)
            quadruples.append((oper[0], left, right, temp))
            quadruples.append(('=', temp, None, left))
        else:
            gen_quad(oper)
```

Directorio de procedimientos metodo gen_temp()

Este método recibe 1 operador, y 2 operandos, regresa la dirección de memoria de una variable temporal que se genera a partir de los datos de entrada. El método empieza obteniendo los tipos de datos de los operandos, para luego generar el tipo de dato del temporal con uso del cubo semántico, se obtiene la dirección del temporal actual, esta se incrementa por 1 en caso de tener más temporales, se agrega la variable temporal y se regresa la dirección generada.

```python
def gen_temp(self, operator:str, operand1:str, operand2:str) -> int:
    """generates a temporal variable

    Args:
        operator (str): operator
        operand1 (str): left operand
        operand2 (str): right operand

    Raises:
        TypeError: Value mismatch

    Returns:
        int: address
    """
    try:
        #print(operator,operand1,operand2)
        var1 = self.get_addr_datatype(operand1)
        var2 = self.get_addr_datatype(operand2)

        #print(operator,var1,var2,operand1,operand2)
        temp_datatype = SemanticCube[operator][var1][var2]
        addr = self.memory['temp']['curr_addr'][temp_datatype]
        self.memory['temp']['curr_addr'][temp_datatype] += 1
        self.add_variable(addr, temp_datatype, addr, 'temp')
        return addr
    except:
        msg = "Error while trying operation: {0} {1} {2}".format(operand2,operator,operand1)
        raise TypeError(msg)
```