Tecnológico de Monterrey

Diseño de compiladores

Documentación Final
MeMyself
MMS11

Héctor Noel León Quiroz
A01251806

25 de noviembre 2020

Índice

Descripción del proyecto

Propósito

Este proyecto fue hecho como trabajo final para la materia de Desarrollo de Compiladores con el objetivo de demostrar lo aprendido en el transcurso de clase al desarrollar desde cero un lenguaje de programación de alto nivel tal como C++, Python, Java etc.

Objetivo

El objetivo deseado es construir un lenguaje de programación que sea capaz de tener un output gráfico. Iniciando el desarrollo desde la especificación de la gramática y la sintaxis hasta llegar a la codificación de la máquina virtual la cual se encarga de ejecutar el código de nuestro lenguaje.

Alcance del proyecto

El lenguaje debe de ser capaz de manejar:

- Estatutos de asignación, condiciones, ciclos, lectura y escritura.
- Expresiones Matemáticas: Aritmética, Lógica y Relacional.
- Módulos, incluyendo parámetros y variables locales y globales.
- Elementos estructurados: Arreglos (una y dos dimensiones), o, listas, etc.

Análisis de requerimientos

Requerimientos Funcionales

- RF01: El lenguaje permitirá declarar variables globales y locales en módulos (funciones), especificando que tipo de variable es al momento que se crean (int, float o char)
- RF02: El lenguaje permitirá declarar variables de arreglos de 1 o 2 dimensiones, estas pueden ser globales o locales en módulos (funciones), especificando que tipo de variable es al momento que se crean (int, float o char)
- RF03: Los arreglos deben ser inicializados al momento de ser declarados con un solo valor int el cual determinará su tamaño.

- RF04: El lenguaje permitirá declarar módulos (funciones), los cuales se tiene que especificar el tipo de retorno ya sea int, float, char o void.
- RF05: El lenguaje permitirá hacer llamadas a módulos (funciones), previamente declarados.
- RF06: Las llamadas a módulos (funciones) deberán tener el número exacto de parámetros con los que fueron declarados o de lo contrario se mostrará un error.
- RF07: Las llamadas a módulos (funciones) de tipo void no podrán ser asignadas a variables, de lo contrario se mostrará un error.
- RF08: Se deberá utilizar una expresión de tipo int para poder accesar los valores de un arreglo.
- RF0: Para acceder a un arreglo se necesita un número mayor igual a 0 y menor al tamaño del arreglo, de lo contrario se mostrará un mensaje de error correspondiente.
- RF09: El lenguaje permitirá mostrar información en la consola por medio del estatuto write.
- RF10: El lenguaje permitirá obtener información mediante la consola por medio del estatuto read.
- RF11: El lenguaje permitirá utilizar llevar a cabo condiciones por medio de los estatutos if y else.
- RF12: El lenguaje permitirá hacer ciclos por medio de los estatutos while y from-to.
- RF13: El lenguaje permitirá regresar un canvas con elementos gráficos con los estatutos Line, Point, Circle, Arc, Penup, Pendown, Color, Size y Clear.

Requerimientos No Funcionales

- RNF01: Se debe de usar un sistema operativo compatible con python
- RNF02: El usuario debe tener instalado python
- RNF03: El usuario debe de tener las dependencias necesarias para compilar y ejecutar el lenguaje
- RNF04 : El lenguaje deberá ser fácil y sencillo de utilizar para nuevos usuarios.
- RNF05: El tiempo de compilación del lenguaje deberá ser aceptable.

Caso de Uso	Crear un programa
ID	UC01
Actores	User
Descripción	Se tecleara un programa el cual es capaz de imprimir “Hello world”

Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself

Flujo de eventos	1. Crear un nuevo archivo
	2. Teclear algo parecido al siguiente código:
	Program UC1;
	
	main(){
	write("Hello world");
	}
	3. Ejectuar tu archivo de la siguiente forma py MeMyself.py { NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola el mensaje Hello world

Caso de Uso	Lectura y escritura
ID	UC02
Actores	User
Descripción	Se tecleara un programa el cual tomara un input del usuario y lo despliegara en pantalla
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself

Flujo de eventos	4. Crear un nuevo archivo
	5. Teclear algo parecido al siguiente código:
	Program UC2;
	var int: x;
	
	main(){
	write("Ingresa un numero");
	read(x);
	write(“El numero ingresado es”, x);
	}
	6. Ejectuar tu archivo de la siguiente forma py MeMyself.py {NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola el todos los cuadrados del 1 al número deseado

Caso de Uso	Ciclos y operaciones aritméticas
ID	UC03
Actores	User
Descripción	Se tecleara un programa el cual es capaz de calcular e imprimir en pantalla la potencia por 2 de 1 al número deseado
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself

Flujo de eventos	7. Crear un nuevo archivo
	8. Teclear algo parecido al siguiente código:
	`Program UC3;`
	`var int: x, i;`
	
	`main(){`
	`write("Ingresa un numero");`
	`read(x);`
	`from i=1 to x+1 do{`
	`write("El cuadrado de ", i, " es ", i*i);`
	`}`
	`}`
	9. Ejectuar tu archivo de la siguiente forma py MeMyself.py {NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola el todos los cuadrados del 1 al número deseado

Caso de Uso	Condicionales y declaración de módulos
ID	UC04
Actores	User
Descripción	Se tecleara un programa el cual es capaz de imprimir un mensaje depende del input del usuario
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself
Flujo de eventos	10. Crear un nuevo archivo

	11. Teclear algo parecido al siguiente código:
	Program UC4;
	var int: x, i;
	char: a;
	
	module void continuar(char c);
	{
	if(c == 'Y') then{
	write("Continua!");
	}
	else {
	write("Adios!");
	}
	}
	
	main(){
	write("Quires proceder? Y/N");
	read(a);
	continuar(a);
	}
	12. Ejectuar tu archivo de la siguiente forma py MeMyself.py { NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola lo correspondiente a lo tecleado

Test Case	Fibonacci Iterativo
ID	TC01
Actores	User
Descripción	Se tecleara un programa el cual es capaz de calcular e imprimir los primeros números deseados la serie de fibonacci
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself
Flujo de eventos	13. Crear un nuevo archivo 14. Teclear algo parecido al siguiente código: Program TC1; var int : num, fib1, fib2, i;

	main(){
	num = 0;
	while(num < 2) do{
	write("Introduce un numero mayor a 0");
	read(num);
	}
	write("Los ", num, " primeros numeros de la serie de Fibonacci son:");
	fib1 = 0;
	fib2 = 1;
	write(fib1);
	from i=2 to num+1 do{
	write(fib2);
	fib2 = fib1 + fib2;
	fib1 = fib2 - fib1;
	}
	}
	15. Ejectuar tu archivo de la siguiente forma py MeMyself.py {NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola la serie de fibonacci

Test Case	Fibonacci Recursivo
ID	TC02
Actores	User
Descripción	Se tecleara un programa el cual es capaz de calcular e imprimir los primeros números deseados la serie de fibonacci
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself

Flujo de eventos	16. Crear un nuevo archivo
	17. Teclear algo parecido al siguiente código:
	Program TC2;
	var int : num, i;
	
	module int fib(int x);
	{

	if((x==1)|(x==0)) then {
	return(x);
	}else {
	return(fib(x-1)+fib(x-2));
	}
	}
	
	main(){
	num = 0;
	while(num < 1) do{
	write("Introduce un numero mayor a 0");
	read(num);
	}
	write("Los ", num, " primeros numeros de la seria de Fibonacci son:");
	from i=0 to num do{
	write(fib(i));
	}
	
	}
	18. Ejectuar tu archivo de la siguiente forma py MeMyself.py {NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en la consola la serie de fibonacci

Test Case	Dibujar circulo
ID	TC03
Actores	User
Descripción	Se tecleara un programa el cual es capaz de dibujar un círculo rojo del tamaño deseado
Precondiciones	Tener los archivos necesarios para correr el lenguaje MeMyself
Flujo de eventos	19. Crear un nuevo archivo 20. Teclear algo parecido al siguiente código: Program TC3; var int : num;

	module void dibujar(int x);
	{
	Size(720,480);
	Pendown(300,200);
	Circle(x, "red");
	Penup();
	}
	
	main(){
	num = 0;
	write("Introduce el tamaño del circulo");
	read(num);
	dibujar(num);
	
	}
	21. Ejectuar tu archivo de la siguiente forma py MeMyself.py {NOMBRE DEL ARCHIVO}

Postcondiciones	Se mostrará en una ventana un círculo

Proceso General Seguido

El proceso seguido de desarrollo fue una metodología ágil de avances semanales los cuales fueron dictados por los profesores de clase, cada semana se entregaba un avance por medio de un archivo zip, el cual contiene los archivos de código junto con un ReadMe el cual describe brevemente el alcance real de la semana.

Bitácora

Fecha	Avances
1 de octubre 2020	Diagramas de Sintaxis y Gramática hechos
8 de octubre 2020	Se tiene el análisis del Léxico y Sintaxis completo (parser y el scanner) en ply. Tuve unos problemas de shift/reduce conflicts con las variables y funciones, el parser no podía detectar cuando termina la declaración de variables y por ende tuve que cambiar el orden de module y tipo al momento de declarar una

	función.
14 de octubre 2020	Se completó la tabla de variables, funciones y el cubo semántico.
24 de octubre 2020	Se completo el código de cuádruplos para expresiones aritméticas básicas
30 de octubre 2020	Se tiene la generación de cuádruplos con direcciones de memoria para expresiones y funciones, falta ciclos y arreglar errores y bugs.
6 de noviembre 2020	Se tiene completo el avance desde la generación de todos los cuádruplos con las direcciones de memoria y la ejecución de operaciones aritméticas en la Máquina Virtual.
13 de noviembre 2020	Se tiene completa toda la generación de código, y toda la ejecución en la máquina virtual excepto las funciones especiales de output gráfico.
21 de noviembre 2020	Se tiene completo todo el compilador y máquina virtual para el lenguaje de programación, se tiene un pequeño avance de la documentación.

Reflexión

Al realizar este proyecto realmente pude entender todo lo que conlleva realizar un lenguaje de programación y todo lo que sucede cuando uso un lenguaje. Esto me ha generado una nueva forma de ver al momento de programar, por ejemplo en la eficiencia del código, ahora que sé todo lo que se genera, con más razón me hace considerar la forma en que codificó. También este proyecto sirvió como una recapitulación de todas mis clases pasadas, por fin pude aplicar gramática y diagramas a un proyecto real, pude comprender su verdadero funcionamiento y uso. En general ha sido una gran experiencia que ha elevado mis conocimientos de programación y simboliza gran parte de mi carrera como ITC.

Descripción del Lenguaje

Nombre del Leguaje

El nombre del leguaje es “MeMyself” este nombre está basado en el nombre del proyecto pre asignado individual. Al momento de correr un programa de “MeMyself” este va a generar un archivo exec.hl.

Descripción general

El lenguaje de programación “MeMyself” es un lenguaje de tipo imperativo, tal como C++, Python o Java. Esto significa que es capaz de hacer cosas muy similares a éstos, con excepción de operaciones muy avanzadas que no son posibles debido al alcance establecido del proyecto. Este lenguaje te permite declarar variables globales las cuales pueden ser accesadas desde cualquier parte del código y mantienen su valor, al momento de declarar variables se necesita especificar su tipo, ya sea int, float o char. A la vez se pueden declarar funciones las cuales son llamadas módulos en este lenguaje, estos deben de tener un tipo ya sea void, int, float, char, en caso de ser void el módulo no puede tener un valor de retorno de lo contrario, el valor de retorno es el mismo a su tipo declarado. Al momento de declarar un módulo también se pueden declarar variables locales que solo podrán ser accesadas por ese módulo y en caso de tener el mismo nombre de una variable global se dará privilegio a la variable local. Cada programa debe de tener una función main en la cual no se pueden declarar variables por lo que es necesario crearlas de manera global preventivamente si se ocupa usar alguna. Al momento de declarar variables globales o locales se pueden declarar arreglos de 1 o 2 dimensiones, los cuales deben ser declarados de la siguiente manera A[2][2], el número representa el tamaño del arreglo, el arreglo podrá ser accedido con un número mayor o igual a 0 y menor al tamaño del arreglo. El lenguaje “MeMyself” también cuenta con los estatutos condicionales if y else, los ciclos while y from-to, funciones de read y write. También cuenta con funciones especiales de output gráfico estas son Line, Point, Circle, Arc, Penup, Pendown, Color, Size y Clear.

Lista de errores

- Error de sintaxis generado por PLY
- Stack overflow al declarar una variable sin memoria disponible.
- Asignación de módulos tipo void.

- Llamada a un módulo no existente.
- Llamada a un módulo que es declarada posteriormente.
- Llamada a un módulo con el número incorrecto de parámetros.
- Llamada a un módulo con parámetros de un tipo distinto al declarado.
- Falta del estatuto return en módulo de diferente tipo de void.
- Creación de un módulo con nombre igual a una variable global.
- Creación de un módulo con nombre main.
- Declaración de una variable con el mismo nombre a una declarada previamente.
- Creación de una variable con algún nombre reservado, ya sea el nombre del programa, main o un módulo declarado.
- Operaciones con variables no declaradas.
- Operaciones con variables sin ningún valor asignado.
- Type mismatch con operaciones y tipos no compatibles. Asignación a una variable con un tipo diferente al de la variable declarada.
- Indexación de arreglo con un número mayor o menor al tamaño.
- Indexación a una variable que no es arreglo.
- Nombrar una variable con alguna de las palabras reservadas del lenguaje.

Descripción del Compilador

Equipo de computo, Lenguaje y Librerías

El lenguaje fue desarrollado en un sistema con las siguientes especificaciones:

- AMD Ryzen 5 1500X Quad-Core Processor 3.5GHz
- 16GB DDR4 RAM
- Windows 10 64 bit
- NVIDIA GeForce GTX 1070

El lenguaje para el desarrollo del código del proyecto fue Python 3.9.0, se utilizo la libreria PLY para el manejo del análisis del Léxico y Sintaxis (parser y el scanner), la cual implementa la las herramientas de Lex y Yacc para ser utilizadas en Python. Por último se usaron librerías estándar de Python tales como sys y fileinput.

Analisis Lexico

Estos fueron las palabras reservadas utilizadas:

Palabras reservadas	Valor
PROGRAM	Program
VAR	var
MODULE	module
MAIN	main
INT	int
FLOAT	float
CHAR	char
VOID	void
RETURN	return
READ	read
WRITE	write
IF	if
THEN	then
ELSE	else
DO	do
WHILE	while
FROM	from
TO	to
LINE	Line
POINT	Point
CIRCLE	Circle
ARC	Arc
PENUP	Penup

PENDOWN	Pendown
COLOR	Color
SIZE	Size
CLEAR	Clear

Estos fueron los tokens utilizados:

Token	Expresión regular o Significado
ID	[a-zA-Z][a-zA-Z_0-9]*
C_INT	\d+
C_FLOAT	\d+\.\d+
C_STRING	\"[^"]+\"
C_CHAR	\'.\'
PLUS	\+
MINUS	\-
DIVIDE	\/
MULTIPLY	*
EQUALS	\=
COMMA	\,
SEMICOLON	\;
COLON	\:
LPAREN	\(
RPAREN	\)
LBRACKET	\{
RBRACKET	\}
LSBRACKET	\[

RSBRACKET	\]
OR	|
GREATER_THAN	\>
LESS_THAN	\<
IS_EQUAL	\==
NOT_EQUAL	\!=
goto	Ir a un cuádruplo
gotof	Si es falso ir a un cuádruplo
gosub	Ir a un módulo
era	Se está llamado a un módulo
Endfunc	Se acabó un módulo
parameter	El parámetro para un módulo
ver	Verificar el índice de un arreglo
writeEnd	Agregar un salto de línea

Análisis de sintaxis

Las gramáticas utilizadas fueron las siguientes:

```
PROGRAMA : PROGRAM ID SEMICOLON VA FU MAIN LPAREN RPAREN BLOQUE
VA : VARS | empty
FU : FUNC FU| empty

VARS : VAR VARSA
VARSA : TIPO COLON ID ARREGLO VARSB
VARSB : COMMA ID ARREGLO VARSB | SEMICOLON VARSA | SEMICOLON
TIPO : INT| FLOAT | CHAR

ARREGLO :  LSBRACKET EXP RSBRACKET ARREGLO2 | empty
ARREGLO2 : LSBRACKET EXP RSBRACKET

ARREGLOCALL : LSBRACKET EXP RSBRACKET ARREGLOC2 | empty
ARREGLOC2 : LSBRACKET EXP RSBRACKET | empty
```

```
TIPOFUNC : INT | FLOAT | CHAR | VOID

FUNC :  MODULE TIPOFUNC ID LPAREN PARAMS RPAREN SEMICOLON FUNCSB
FUNCSB : VARS BLOQUE | BLOQUE

PARAMS : PARAMSL | empty
PARAMSL : TIPO ID | PARAMSL COMMA TIPO ID

BLOQUE : LBRACKET BLOQUEA | LBRACKET RBRACKET
BLOQUEA : ESTATUTO BLOQUEA | ESTATUTO RBRACKET

ESTATUTO : ASIGNACION
      | LLAMADA
      | RETORNO
      | LECTURA
      | ESCRITURA
      | DECISION
      | REPETICION
      | FUNCESP

ASIGNACION : ID ARREGLOCALL EQUALS EXPRESION SEMICOLON

LLAMADA : ID LPAREN LLAMADAS
      | ID LPAREN RPAREN SEMICOLON
LLAMADAS : EXPRESION RPAREN SEMICOLON
      | EXPRESION COMMA LLAMADAS

RETORNO : RETURN LPAREN EXPRESION RPAREN SEMICOLON

LECTURA : READ LPAREN LECTURAS
LECTURAS : ID COMMA LECTURAS
      | ID RPAREN SEMICOLON

ESCRITURA : WRITE LPAREN ESCRITURAA
ESCRITURAA : C_STRING COMMA ESCRITURAA
      | C_STRING RPAREN SEMICOLON
      | EXPRESION COMMA ESCRITURAA
      | EXPRESION RPAREN SEMICOLON

DECISION : IF LPAREN EXPRESION RPAREN THEN BLOQUE SINO
SINO : ELSE BLOQUE | empty
```

```
EXPRESION : EXP
      | EXP GREATER_THAN EXPRESION
      | EXP LESS_THAN EXPRESION
      | EXP NOT_EQUAL EXPRESION
      | EXP AND EXPRESION
      | EXP OR EXPRESION
      | EXP IS_EQUAL EXPRESION

EXP : TERMINO
| TERMINO PLUS EXP
| TERMINO MINUS EXP

TERMINO : FACTOR
      | FACTOR MULTIPLY TERMINO
      | FACTOR DIVIDE TERMINO

FACTOR : LPAREN EXPRESION RPAREN
      | VARCTE
      | LLAMADAMOD

LLAMADAMOD : ID LPAREN LLAMADAMODS
      | ID LPAREN RPAREN
LLAMADAMODS : EXPRESION RPAREN
            | EXPRESION COMMA LLAMADAMODS

VARCTE : ID pushPilaOp ARREGLOCALL
      | C_INT
      | C_FLOAT
      | C_CHAR

FUNCESP : FLINE
      | FPOINT
      | FCIRCLE
      | FARC
      | FPENUP
      | FPENDOWN
      | FCOLOR
      | FSIZE
      | FCLEAR

FLINE : LINE LPAREN EXPRESION COMMA EXPRESION RPAREN SEMICOLON

FILLCOLOR : COMMA C_STRING |  empty
```

```
FPOINT : POINT LPAREN EXPRESION FILLCOLOR RPAREN SEMICOLON

FCIRCLE : CIRCLE LPAREN EXPRESION FILLCOLOR RPAREN SEMICOLON

FARC : ARC LPAREN EXPRESION COMMA EXPRESION FILLCOLOR RPAREN
      SEMICOLON

FPENUP : PENUP LPAREN RPAREN SEMICOLON

FPENDOWN : PENDOWN LPAREN EXPRESION COMMA EXPRESION RPAREN
           SEMICOLON

FCOLOR : COLOR LPAREN C_STRING RPAREN SEMICOLON

FSIZE : SIZE LPAREN EXPRESION COMMA EXPRESION RPAREN SEMICOLON

FCLEAR : CLEAR LPAREN RPAREN SEMICOLON

empty :
```

Generación de Código Intermedio y Análisis Semántico

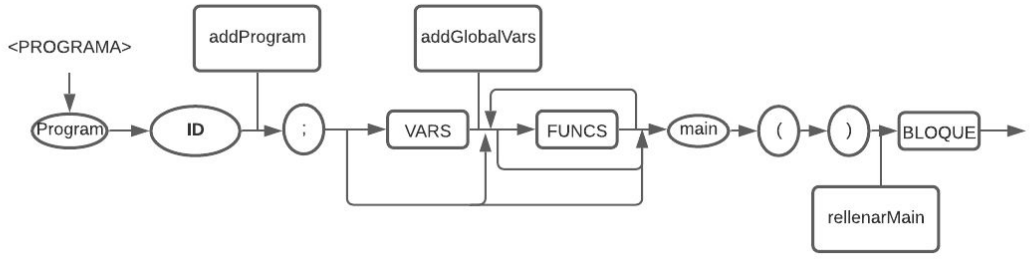
Código de Operación y Direcciones Virtuales

Las direcciones virtuales que se utilizaron para las variables fueron las siguientes:

Variable	Tipo	Rango
Global	int	2000 - 4999
	float	5000 - 7999
	char	8000 - 10999
Local	int	11000 - 13999
	float	14000 - 16999
	char	17000 - 19999
	int	20000 - 22999

Temporal	float	23000 - 25999
	char	26000 - 28999
	bool	29000 - 31999
Constante	int	32000 - 34999
	float	35000 - 37999
	char	38000 - 40999
	string	41000 - 43999

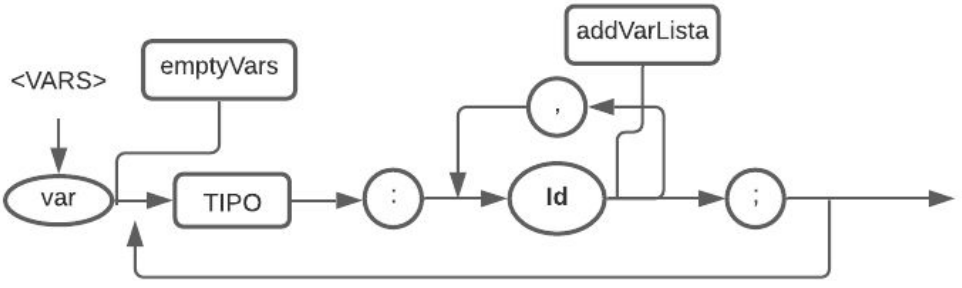
Diagramas de Sintaxis con Acciones



addProgram: Crea un nuevo cuádruplo “goto”, agrega el nombre del programa al diccionario de variables globales.

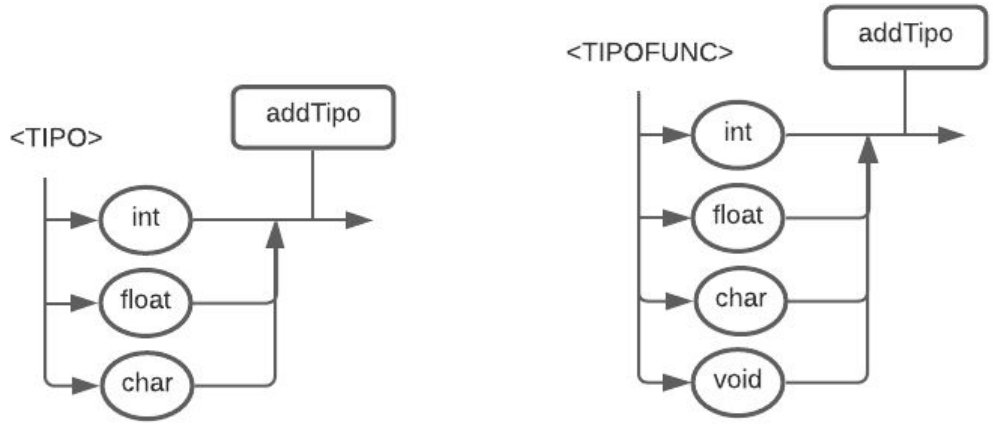
addGlobalVars: Asigna direcciones a todas las variables en “listaVariables” y las agrega al diccionario de variables globales.

rellenarMain: Agrega el número de cuádruplo al “goto” que se agregó en addProgram.

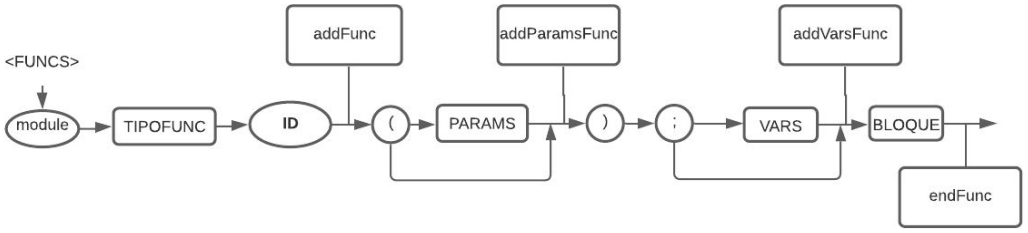


emptyVars: Borra todos los elementos de la lista “listaVariables”.

addVarLista: Agrega el id y tipo de una variable a la lista “listaVariables”.



addTipo: Se guarda el último tipo leido a la variable tipoVar o tipoFunc.

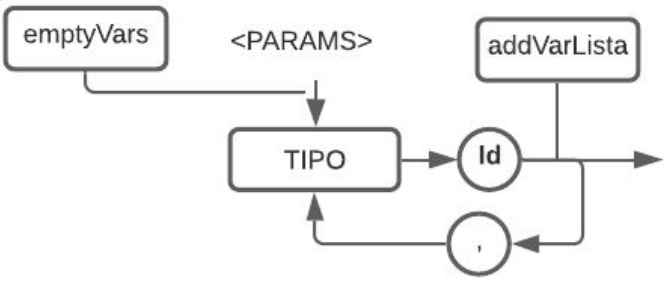


addFunc: Guarda el id de la función en el diccionario de funciones y agrega una nueva variable con el id de la funciona al diccionario de variables globales.

addParamsFunc: Agrega los parámetros guardados en la lista “listaVariables”al diccionario de parámetros dentro del diccionario de la función.

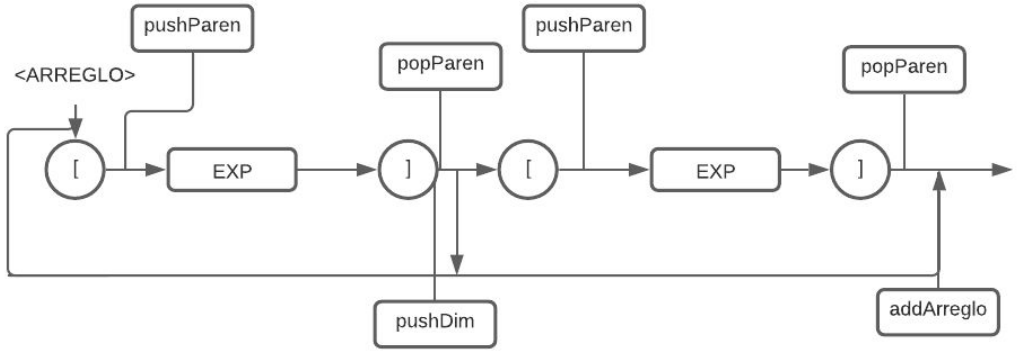
addVarsFunc: Agrega las variables guardadas en la lista “listaVariables”al diccionario de variables locales dentro del diccionario de la función.

endFunc: Agrega el cuádruplo Endfunc y resetea las direcciones locales y temporales.



emptyVars: Borra todos los elementos de la lista “listaVariables”.

addVarLista: Agrega el id y tipo de una variable a la lista “listaVariables”.

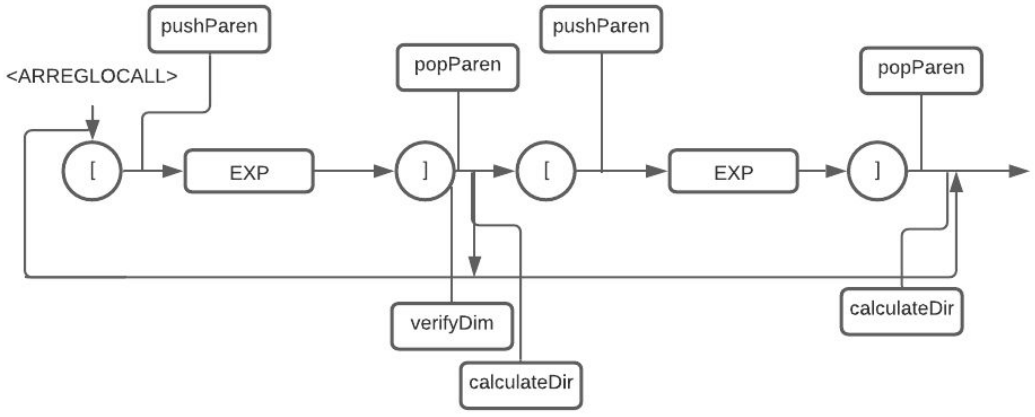


pushParen: Agrega ‘(‘ a pilaPoper

popParen: Saca ‘(‘ de pilaPoper si es el elemento en el tope.

pushDim: Saca el valor de la dimensión de pilaOp y lo agrega a pilaDim

addArreglo: Declara la variable como arreglo en el diccionario de variables y agrega las dimensiones de pilaDim

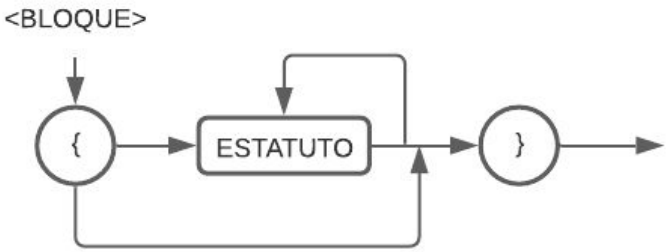


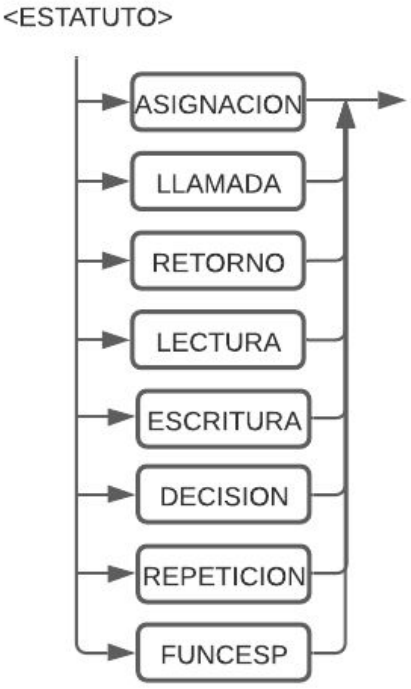
pushParen: Agrega ‘(‘ a pilaPoper

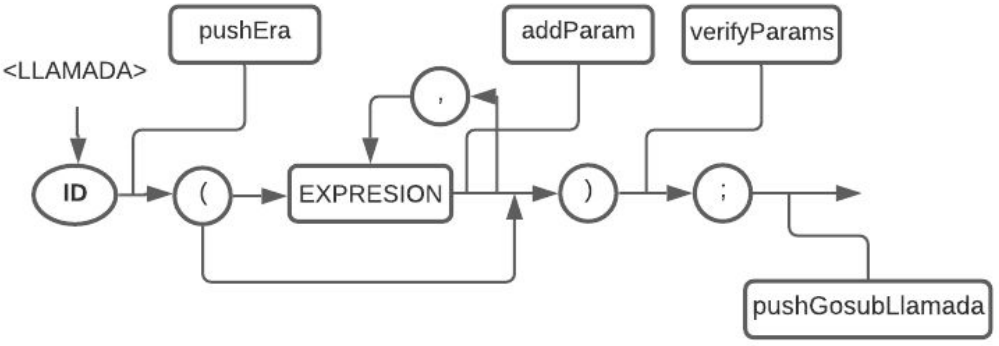
popParen: Saca ‘(‘ de pilaPoper si es el elemento en el tope.

verifyDim: Agrega el cuádruplo “ver” que verifica que el índice deseado esté entre las dimensiones del arreglo.

calculateDir: Agrega los cuádruplos necesarios para calcular la dirección del índice que se pide.





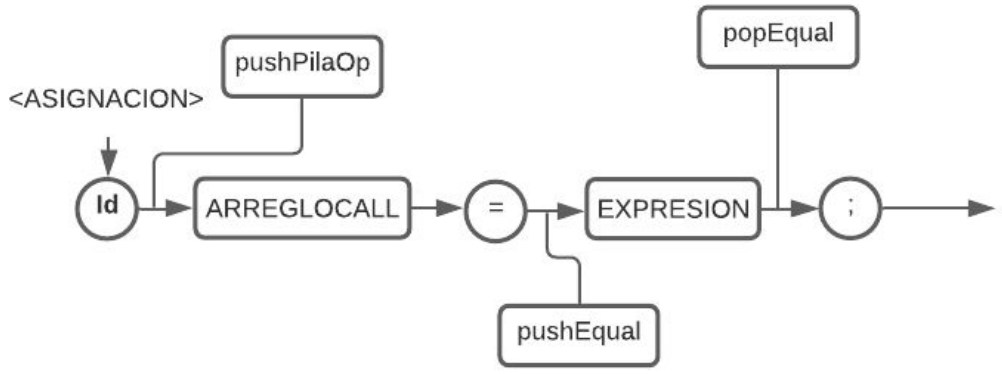


pushEra: Agrega el id a pilaLlamadas y crea el cuádruplo, "era" con el nombre de la función.

addParam: Se guarda el parámetro y su tipo en pilaParams.

verifyParams: Verifica que los parámetros en pilaParams sean la cantidad y el tipo correcto, y luego crea un cuádruplo “parameter” para cada uno.

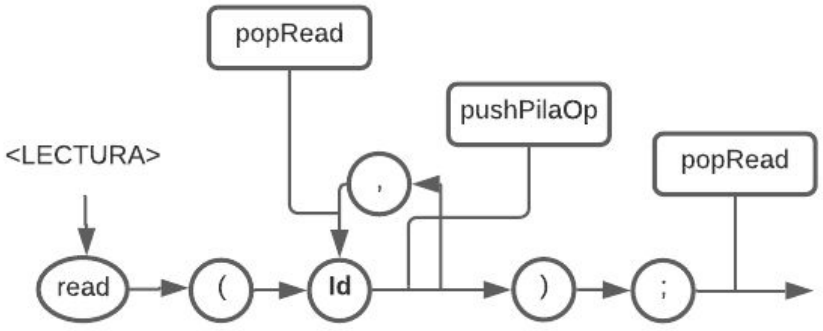
pushGosubLlamada: Crea el cuádruplo “gosub” con el nombre de la función.



pushPilaOp: Agrega la id a pilaOp y el tipo a pilaTipo.

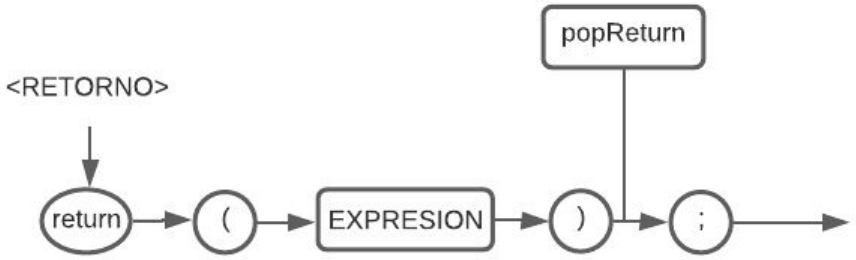
pushEqual: Agrega ‘=’ a pilaPoper.

popEqual: Si el tope de pilaPoper es ‘=’ y los tipos son compatibles para los dos siguientes valores de pilaOp se crea el cuádruplo “=”.

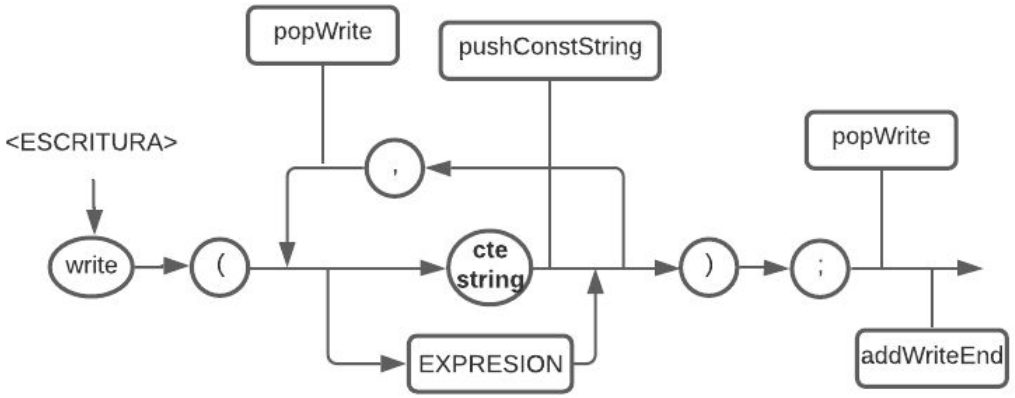


pushPilaOp: Agrega la id a pilaOp y el tipo a pilaTipo.

popRead: Se saca la id y el tipo de pilaOp y pilaTipo y se crea el cuádruplo “read”.



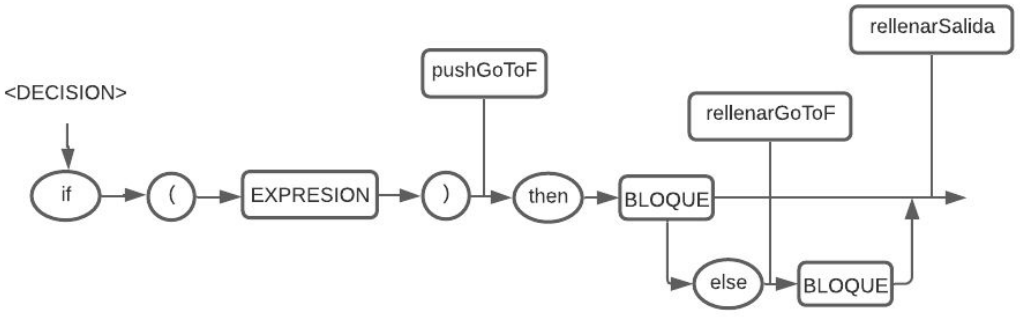
popReturn: Verifica el tipo de retorno de la función, crea el cuádruplo "return", crea el cuádruplo "goto" y agrega el número de cuádruplo a la pilaReturn.



pushConstString: Se agrega la constante a la tabla de constantes si es que no existe, luego se guarda el valor y el tipo en pilaOp y pilaTipo.

popWrite: Se saca la id y el tipo de pilaOp y pilaTipo y se crea el cuádruplo “write”.

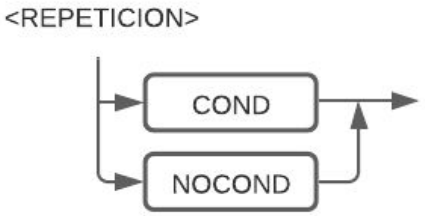
addWriteEnd: Se crea el cuadruplo “writeEnd”.

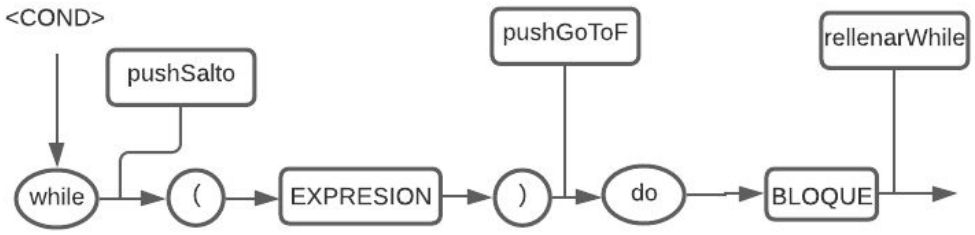


pushGoToF: Verifica que el tipo del tope sea bool, crea el cuádruplo “gotof” y agrega el número del cuádruplo a pilaSaltos.

rellenarGoToF: Crea el cuádruplo "goto", saca el elemento de la pilaSaltos y rellena el cuádruplo "gotof" que apunta con el siguiente número del cuádruplo, por último agrega el número de cuádruplo actual a la pilaSaltos.

rellenarSalida: Saca el elemento de la pilaSaltos y rellena el cuádruplo "goto" que apunta con el siguiente número del cuádruplo.

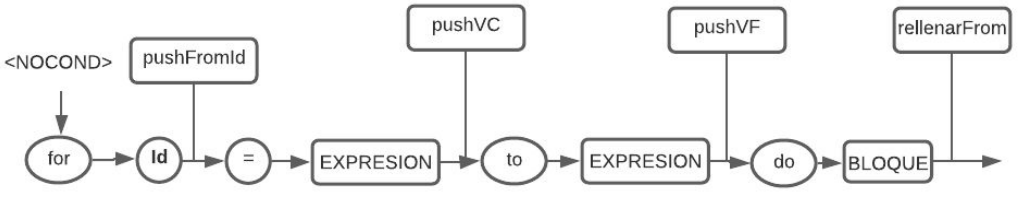




pushSalto: Agrega el número de cuádruplo actual a pilaSaltos.

pushGoToF: Verifica que el tipo del tope sea bool, crea el cuádruplo "gotof" y agrega el número del cuádruplo a pilaSaltos.

rellenarWhile: Crea el cuádruplo "goto" con el número agregado en pushSalto y rellena el "gotof" de pushGoToF.

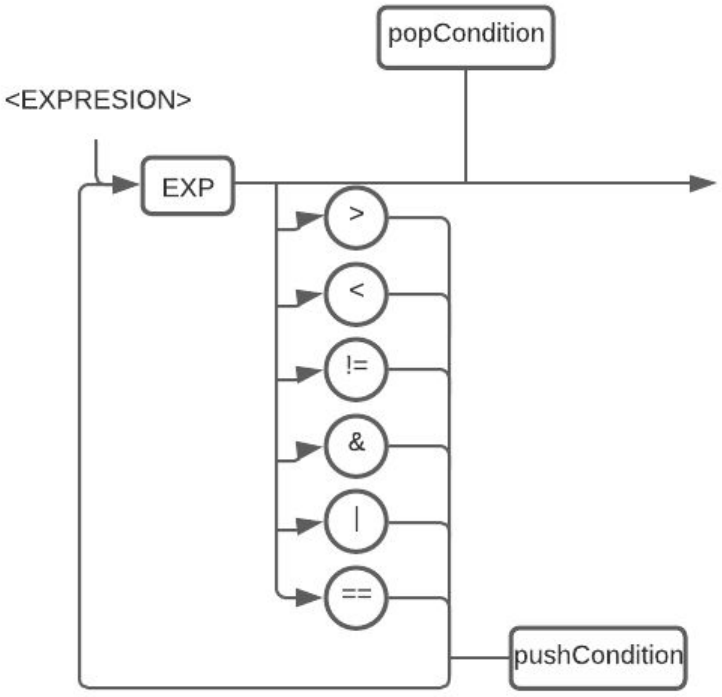


pushFromId: Verifica la variable y agrega la id a pilaOp y el tipo a pilaTipo.

pushVC: Verifica el tipo de la constante con la variable y agrega un cuádruplo "=".

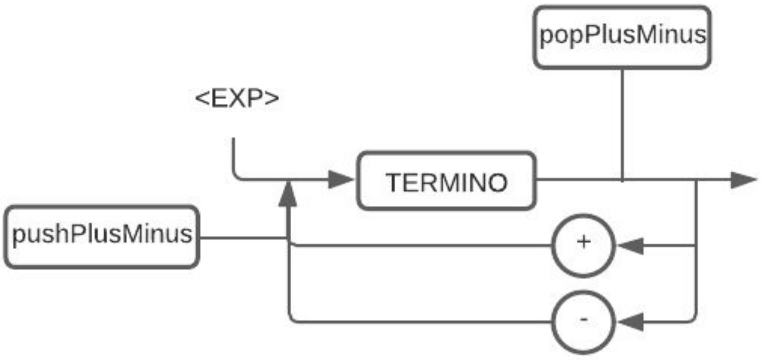
pushVF: Agrega el cuádruplo "<" para comparar VC con VF y agrega el número del cuádruplo a pilaSaltos, luego agrega el cuádruplo "gotof" y agrega el número del cuádruplo a pilaSaltos.

rellearFrom: Agrega el cuádruplo "+" para sumar VC más 1, rellena el "gotof" de pushVF y crea el cuádruplo "goto" con el número agregado en pushVF.



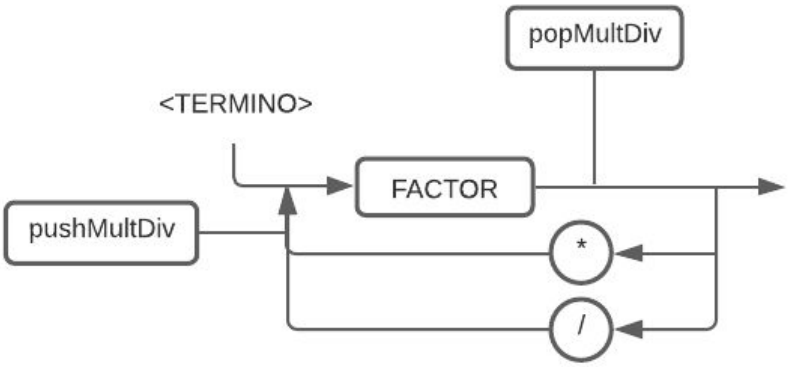
pushCondtion: Agrega la condición a pilaPoper.

popCondition: Si el tope de pilaPoper es una condición y los tipos son compatibles para los dos siguientes valores de pilaOp se crea el cuádruplo.



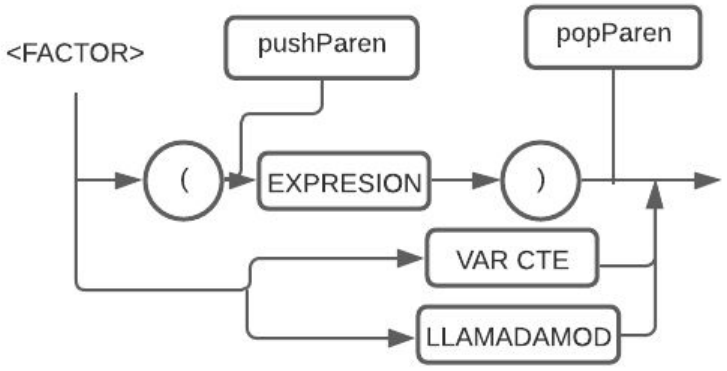
pushPlusMinus: Agrega la '+' o '-' a pilaPoper.

popPlusMinus: Si el tope de pilaPoper es '+' o '-' y los tipos son compatibles para los dos siguientes valores de pilaOp se crea el cuádruplo.



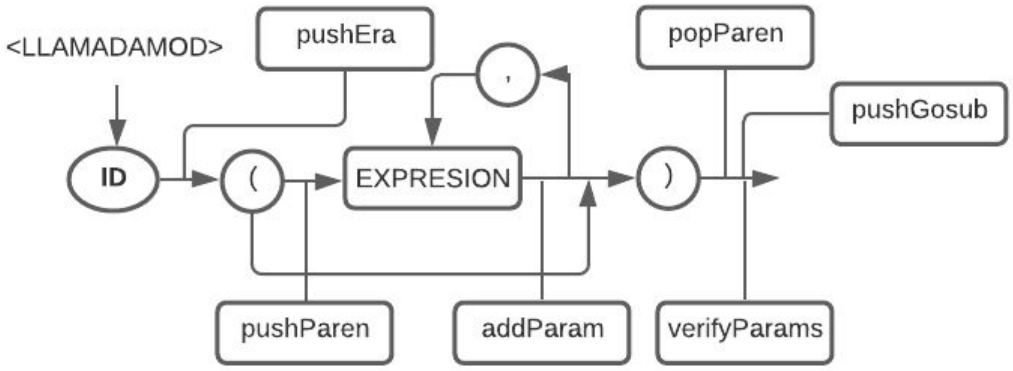
pushMultDiv: Agrega la '*' o '/' a pilaPoper.

popMultDiv: Si el tope de pilaPoper es '*' o '/' y los tipos son compatibles para los dos siguientes valores de pilaOp se crea el cuádruplo.



pushParen: Agrega '(' a pilaPoper

popParen: Saca '(' de pilaPoper si es el elemento en el tope.



pushEra: Agrega el id a pilaLlamadas y crea el cuádruplo, "era" con el nombre de la función.

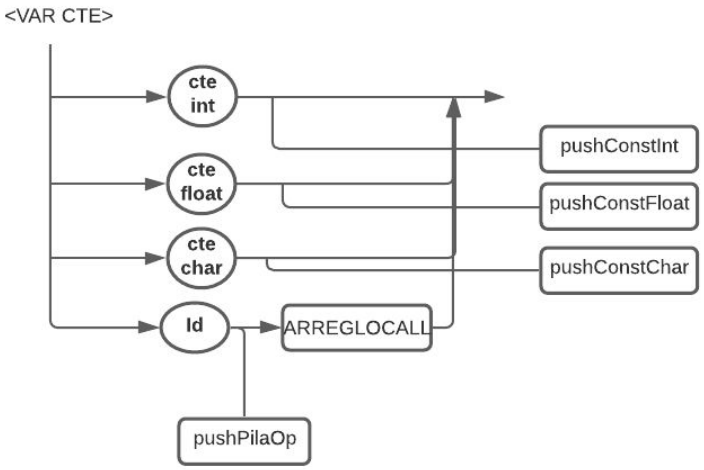
pushParen: Agrega '(' a pilaPoper

addParam: Se guarda el parámetro y su tipo en pilaParams.

popParen: Saca '(' de pilaPoper si es el elemento en el tope.

verifyParams: Verifica que los parámetros en pilaParams sean la cantidad y el tipo correcto, y luego crea un cuádruplo "parameter" para cada uno.

pushGosub: Crea el cuádruplo "gosub" con el nombre de la función y crea el cuádruplo '=' para la variable global de la función.

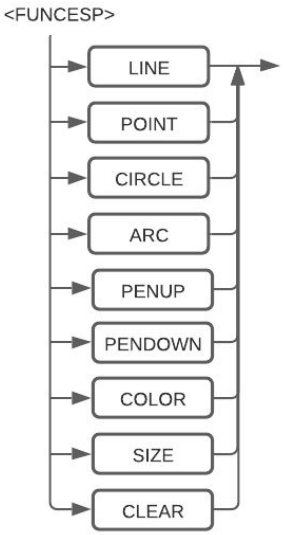


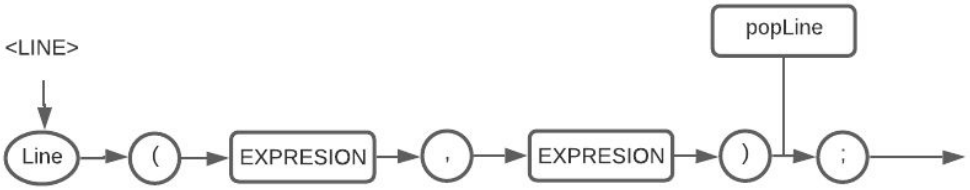
pushPilaOp: Agrega la id a pilaOp y el tipo a pilaTipo.

pushConstInt: Se agrega la constante a la tabla de constantes si es que no existe, luego se guarda el valor y el tipo en pilaOp y pilaTipo.

pushConstFloat: Se agrega la constante a la tabla de constantes si es que no existe, luego se guarda el valor y el tipo en pilaOp y pilaTipo.

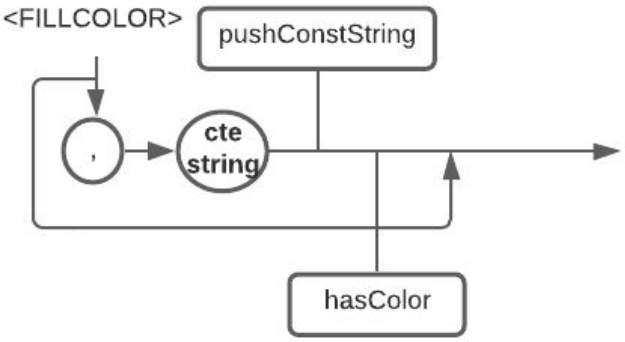
pushConstChar: Se agrega la constante a la tabla de constantes si es que no existe, luego se guarda el valor y el tipo en pilaOp y pilaTipo.





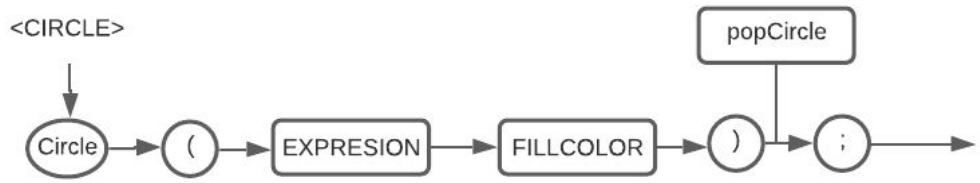
popLine: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo “line”.

popPoint: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo “point”.



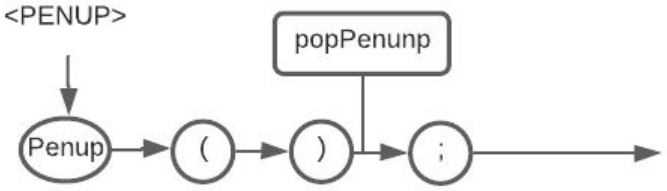
pushConstString: Se agrega la constante a la tabla de constantes si es que no existe, luego se guarda el valor y el tipo en pilaOp y pilaTipo.

hasColor: La variable hasColor = true

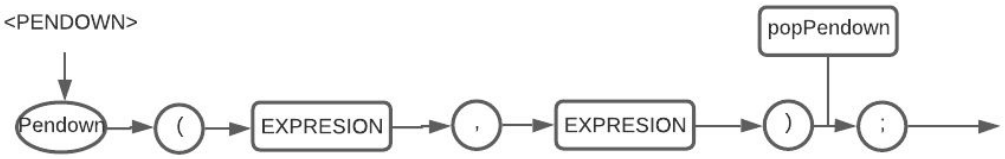


popCircle: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo "point".

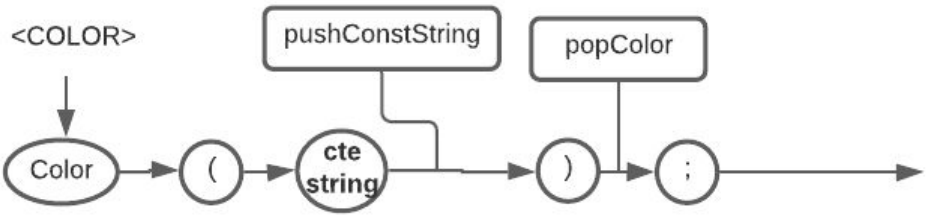
popArc: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo "arc".



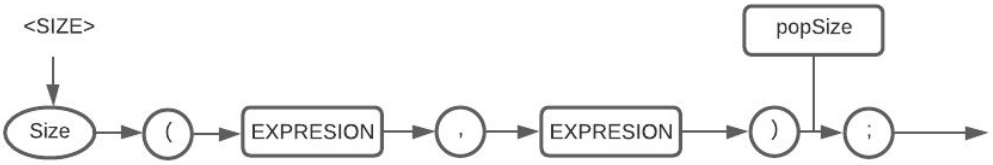
popPenup: Se crea el cuádruplo "penup".



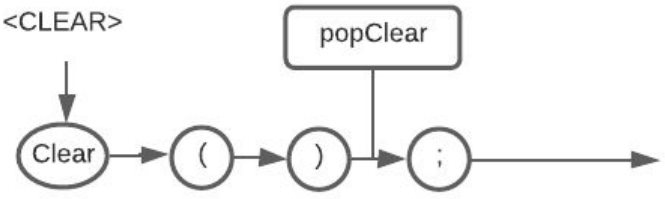
popPendown: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo "pendown".



popColor: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo "color".



popSize: Se saca las id y los tipos de pilaOp y pilaTipo, se crea el cuádruplo "size".



popClear: Se crea el cuádruplo "clear".

Tabla de consideraciones semánticas

LTipo	RTipo	+	-	*	/	>	<	==	!=	&	|	=
Int	int	int	int	int	int	bool	bool	bool	bool	error	error	int
	float	float	float	float	float	bool	bool	bool	bool	error	error	error
	char	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	error	error	error	error	error	error	error
Float	int	float	float	float	float	bool	bool	bool	bool	error	error	error
	float	float	float	float	float	bool	bool	bool	bool	error	error	float
	char	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	error	error	error	error	error	error	error
Char	int	error	error	error	error	error	error	error	error	error	error	error
	float	error	error	error	error	error	error	error	error	error	error	error
	char	error	error	error	error	error	error	bool	bool	error	error	char
	bool	error	error	error	error	error	error	error	error	error	error	error
Bool	int	error	error	error	error	error	error	error	error	error	error	error
	float	error	error	error	error	error	error	error	error	error	error	error
	char	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	error	error	bool	bool	bool	bool	bool

Proceso de Administración de Memoria

Tabla de Variables

```
self.varDict[name] = {
            "type": String,
            "dir": Int,
            "isArray": Bool,
            "dim1": Int,
            "dim2": Int
        }
```

La tabla de variables es una clase con un directorio. La llave es el nombre de la variable y los atributos son los siguientes.

La clase contiene métodos para agregar variables, ver si una variable existe, buscar variable por nombre, buscar variable por dirección, e imprimir todas las variables.

El compilador crea un objeto tipo TablaVariables para las variables globales.

Tabla de Funciones

```
self.funcDict[name] = {
  "returnType": String,
  "varTable": TablaVariables(),
  "paramTable": TablaVariables(),
  "numberParams" : Int,
  "cuadNumber": Int,
  "dirGlobal": int
}
```

La tabla de funciones es una clase con un directorio. La llave es el nombre de la variable y los atributos son los siguientes.

La clase contiene métodos para agregar funciones, ver si una función existe, conseguir una función, agregar variables locales, buscar variables o parámetros, agregar parámetros, imprimir todas las funciones y exportar la tabla a un archivo.

El compilador crea un objeto tipo TablaFunciones.

Tabla de Constantes

```
TablaVariables.__init__(self)
```

La tabla de constantes es una clase que hereda de TablaVariables.

La clase contiene los mismos atributos y métodos que TablaVariables, con un método extra para exportar las constantes a un archivo.

El compilador crea un objeto tipo TablaConstantes.

Direcciones

```
self.start = Int
self.end = Int
self.actual = Int



Direccion = {
"globalint" : Direcciones(2000, 4999)
….
}
```

Direcciones es una clase con los siguientes atributos.
Tiene métodos para conseguir una nueva dirección, incrementar el número de direcciones, conseguir la cantidad de direcciones asignadas y resetear el contador.

El compilador crea un objeto de Dirección por cada contexto y tipo.

Cuádruplos

```
self.stack = []
self.count = 0









0 ) Cuadruplo: goto -1 -1 49
1 ) Cuadruplo: * 11000 32002 20000
2 ) Cuadruplo: - 2002 20000 20001
3 ) Cuadruplo: + 20001 11000 20002
4 ) Cuadruplo: + 11000 20002 20003
5 ) Cuadruplo: = 20003 -1 11001
6 ) Cuadruplo: == 11000 32003 29000
```

Cuadruplos es una clase con los siguientes atributos.

La clase contiene métodos para agregar cuádruplos, rellenar un cuádruplo, contar los cuádruplos, imprimir todas los cuádruplos y exportar los cuádruplos archivo.

El compilador crea un objeto tipo Cuadruplos y se exportan de la siguiente manera.

Cubo Semántico

```
self.cuboSem = {
"int" : {
    "int" : {
        "+" : "int",
        "-" : "int",
        "*" : "int",
        "/" : "float",
        "==" : "bool",
        "!=" : "bool",
        ">" : "bool",
        "<" : "bool",
        "|" : "error",
        "&" : "error",
        "=" : "int"
    }
...
```

Cubo Semántico es una clase con el siguiente diccionario.

La clase contiene un método para conseguir la semántica de una operación.

El compilador crea un objeto tipo CuboSemántico.

Descripción de la Máquina Virtual

Equipo de computo, Lenguaje y Librerías

El equipo de cómputo y el lenguaje de programación que se utilizaron para la Máquina Virtual fueron los mismos que el compilador. Para poder crear los outputs gráficos se requirió el uso de la librería tkinter la cual permitió crear un canvas capaz de mostrar círculos, líneas y arcos.

Proceso de Administración de Memoria en ejecución

La memoria en la máquina virtual se administra con la ayuda de una clase llamada Memoria, la cual es la responsable de la asociación de las direcciones virtuales con las de ejecución, esta clase contiene un simple diccionario donde la llave es la dirección de memoria.

```
self.memoria = {}
Ejemplo:
   memoria['2000'] = 5
```

La clase contiene métodos para asignar una dirección y conseguir el valor de una dirección. Se crearon dos objetos de esta clase en el código de la Máquina Virtual, uno para las variables constantes y otra para las globales. Para manejar las variables locales y temporales se crearon dos pilas de objetos Memoria(), las cuales se les agrega un objeto nuevo al momento de entrar a un contexto de ejecución de una función, y se remueve el objeto una vez que termina la ejecución, así es posible manejar las llamadas recursivas de funciones. Para que la Máquina Virtual sea capaz de ingresar y asignar los valores correctos se crearon funciones las cuales revisan en que rango se encuentra la dirección y se accede al diccionario correspondiente.

Pruebas del Funcionamiento del Lenguaje

Fibonacci Iterativo

```
Program TC1;
var int : num, fib1, fib2, i;

main(){
    num = 0;
    while(num < 2) do{
        write("Introduce un numero mayor a 0");
        read(num);
    }
    write("Los ", num, " primeros numeros de la seria de Fibonacci
son:");
    fib1 = 0;
    fib2 = 1;
    write(fib1);
    from i=2 to num+1 do{
        write(fib2);
        fib2 = fib1 + fib2;
        fib1 = fib2 - fib1;
    }
}
```

```
PS C:\Users\hecto\Documents\Compis> py MeMyself.py c:\Users\hecto\Documents\Compis\TestCases\TC1.txt
Introduce un numero mayor a 0
5
Los 5 primeros numeros de la seria de Fibonacci son:
0
1
1
2
3
```

Fibonacci Recursivo

```
Program TC2;
var int : num, i;



module int fib(int x);
{
   if((x==1)|(x==0)) then {
      return(x);
   }else {
      return(fib(x-1)+fib(x-2));
   }
}

main(){
    num = 0;
    while(num < 1) do{
        write("Introduce un numero mayor a 0");
        read(num);
    }
    write("Los~ ", num, " primeros numeros de la seria de Fibonacci
son:");
    from i=0 to num do{
        write(fib(i));
    }

}
```

```
PS C:\Users\hecto\Documents\Compis> py MeMyself.py c:\Users\hecto\Documents\Compis\TestCases\TC2.txt
Introduce un numero mayor a 0
10
Los 10 primeros numeros de la seria de Fibonacci son:
0
1
1
2
3
5
8
13
21
34
```

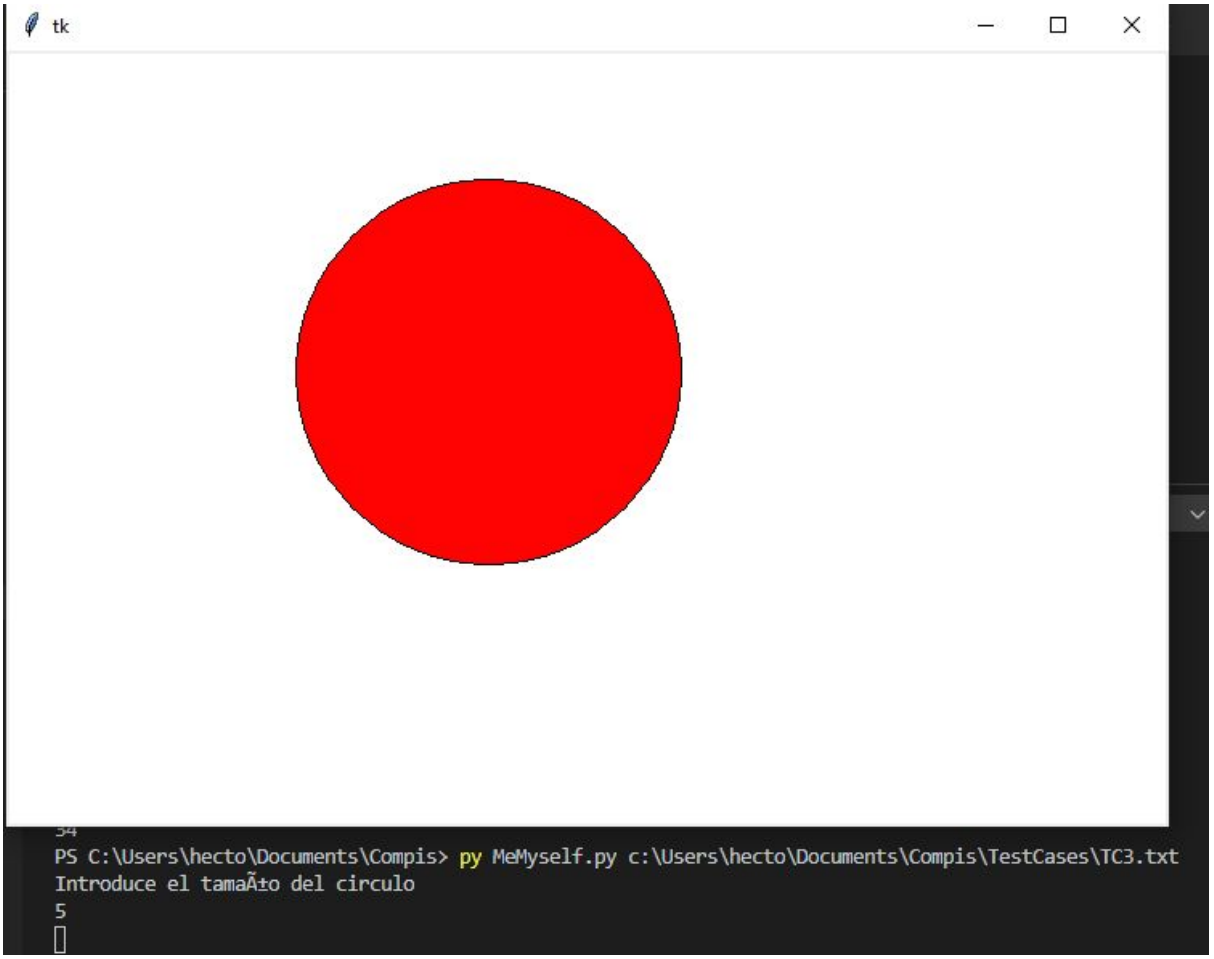
Dibujar Circulo

```
Program TC3;
var int : num;


module void dibujar(int x);
{
    Size(720,480);
    Pendown(300,200);
    Circle(x, "red");
    Penup();
}

main(){
    num = 0;
    write("Introduce el tamaño del circulo");
    read(num);
    dibujar(num);

}
```



Factorial Recursivo

```
Program TC4;
var int: num;

module int fact (int j);
{
      if(j == 1) then
      {
            return (j);}
      else
      {
            return (j * fact(j-1));
      }
}

main(){
      write("Ingresa el factorial a calcular");
      read(num);
```

```
      write(fact(num));
}
```

```
PS C:\Users\hecto\Documents\Compis> py MeMyself.py c:\Users\hecto\Documents\Compis\TestCases\TC4.txt
Ingresa el factorial a calcular
5
120
```

Find Arreglo

```
Program TC5;
var int : num, result, i, Array[20];


module int find(int x);
{
    from i=0 to 20 do{
        if(Array[i] == x) then{
            return(i);
        }
    }
    return(100);
}

main(){
    num = 0;
    from i=0 to 20 do{
        Array[i] = i*2+1;
    }
    write("Que numero quieres buscar?");
    read(num);
    result = find(num);
    if(result < 100) then{
        write("El numero ",Array[result], " se encuentra en el indice
", result);
    }
    else{
        write("El numero ",num," no se encuentra en el arreglo");
    }

}
```

```
PS C:\Users\hecto\Documents\Compis> py MeMyself.py c:\Users\hecto\Documents\Compis\TestCases\TC5.txt
Que numero quieres buscar?
5
El numero 5 se encuentra en el indice 2
```

Multiplicación de Matriz

```
Program TC6;

module void multiplyMat();
var int : m, n, p, q, c, d, k, sum, input ;
    int : first[10][10], second[10][10], multiply[10][10];
{
    sum = 0;
    write("Ingresa el numero de columnas y renglones de la primera
matriz (max 10)");
    read(m);
    read(n);
    write("Ingresa los elementos de la primer matriz");
    from c = 0 to m do{
        from d = 0 to n do{
            read(input);
            first[c][d] = input;
        }
    }
    write("Ingresa el numero de columnas y renglones de la segunda
matriz (max 10)");
    read(p);
    read(q);
    if (n != p) then{
        write("La multiplicacion no es posible");
    }
    else{
        write("Ingresa los elementos de la segunda matriz");
        from c = 0 to p do{
            from d = 0  to q do{
                read(input);
                second[c][d] = input;
            }
        }

        from c = 0 to m do{
            from d = 0  to q do{
                from k = 0 to p do{
```

```
                    sum = sum + first[c][k]*second[k][d];
                }
                multiply[c][d] = sum;
                sum = 0;
            }
        }

        write("Producto de las matrices: ");

        from c = 0 to m do{
            from d = 0  to q do{
                write(multiply[c][d]);
            }
        }
    }
}

main(){
    multiplyMat();
}
```

```
PS C:\Users\hecto\Documents\Compis> py MeMyself.py c:\Users\hecto\Documents\Compis\TestCases\TC6.txt
Ingresa el numero de columnas y renglones de la primera matriz (max 10)
3
3
Ingresa los elementos de la primer matriz
1
2
0
0
1
1
2
0
1
Ingresa el numero de columnas y renglones de la segunda matriz (max 10)
3
3
Ingresa los elementos de la segunda matriz
1
1
2
2
1
1
1
2
1
Producto de las matrices:
5
3
4
3
3
2
3
4
5
```