MeMySelf

Naomi Macias Honti A01282098

Naomi Macias Honti

24 nov. 20

Contents

Descripción del Proyecto

Propósito y Alcance del Proyecto

El propósito de este proyecto es crear un compilador capaz de compilar y ejecutar el lenguaje definido como MeMySelf, especificado en este documento. El programa será capaz de correr funciones, manejar funciones locales y globales, utilizar estatutos de decisión, de repetición condicional y no condicional, de lectura y de escritura y mostrar un output gráfico.

Análisis de Requerimientos y Casos de Uso Generales

RF001. El usuario debe de poder ejecutar el programa.
- RNF001. El programa debe de seguir la estructura definida.
- RNF002. El programa debe de sacar error ante cualquier inconsistencia.

RF002. El usuario debe de poder ingresar datos si la instrucción lo requiere.
- RNF003. Los datos ingresados se deben de guardar en una variable del mismo tipo que el valor esperado.
- RNF004. El programa debe de sacar un error en caso de que el valor ingresado sea incompatible con el tipo de variable.

RF003. El usuario debe de poder desplegar el valor de las variables en la pantalla.
- RNF005. El valor por desplegar puede ser el resultado de una ecuación, un string o el valor de una variable.
- RNF006. La variable por desplegar debe de estar definido e inicializado con algún valor antes de desplegarlo en la pantalla.

RF004. El usuario debe de poder declarar variables del tipo: int, float y char.
- RNF007. El programa debe de también poder manejar variables del tipo booleano y strings.
- RNF008. Las variables declaradas se identifican con nombres únicos.

RF005. El usuario debe de poder declarar funciones del tipo: void, int, float y char.
- RNF009. El programa no debe de dar error si no hay funciones declaradas.
- RNF010. Las funciones pueden nunca ser ejecutadas.
- RNF011. Las funciones pueden no tener instrucciones.

RF006. El usuario debe de obligatoriamente declarar la función main con el código inicial a ejecutar.

RF007. El usuario debe de poder llamar a las funciones declaradas, excepto al main, el cual solo se ejecuta una vez.
- RNF012. El programa debe de comprobar que la función llamada exista.
- RNF013. El programa debe de poder manejar llamadas recursivas.
- RNF014. El programa debe de comprobar que la cantidad de parámetros enviados con la llamada y el tipo de los parámetros enviados sean iguales a los definidos en la función.

RF008. El usuario debe de poder llamar a las funciones determinadas para dibujar en el output gráfico.
- RNF015. El programa no desplegara el output grafico si el usuario no hace llamadas al output gráfico.
- RNF016. El output grafico no debe cerrarse hasta que el usuario lo indique.

RF009. El usuario debe de poder crear instrucciones de decisión y loops.
RF010. El usuario debe de poder declarar variables locales para cada función.
- RNF017. Las variables locales no se repiten entre los parámetros ni las variables globales.
- RNF018. Las variables globales pueden ser utilizadas desde cualquier lugar.
- RNF019. Las variables locales solo pueden ser utilizadas dentro de las funciones correspondientes.

RF011. El usuario debe de poder declarar parámetros para cada función.
- RNF020. Los parámetros deben de ser del tipo de variables aceptados.
- RNF021. Las funciones pueden ser declaradas sin parámetros.

RF012. El usuario debe de poder hacer operaciones con las variables numéricas.
RF013. El usuario debe de poder utilizar constantes numéricas, llamadas a funciones y variables en las expresiones.
- RNF022. Las llamadas a funciones en expresiones deben de regresar un valor numérico.
- RNF023. Las variables en expresiones deben de ser de tipo numérico.

RF014. El usuario debe de poder asignar valores a variables existentes.
- RNF024. El valor asignado a una variable debe de ser compatible con el tipo de la variable.

Descripción de los principales Test Cases

Test Cases que no dan error

ID	Test	Código	Consola Input	Consola Output

1	Program	Program MeMySelf;
		
		main()
		{
		write("Hello World");
		}
				Hello World

2	Write and Read	Program MeMySelf;
		var
		int x;
		char y;
		
		main()
		{
		read(x,y);
		write(x,y,"Hello World",4+1,x*3);
		}
			2
			c
				2
				c
				Hello World
				5
				6

3.1	Call	Program MeMySelf; var int x; char y; module void funcion();	1 c	1 c Hello World

		{
		read(x,y);
		}
		
		main()
		{
		funcion();
		write(x,y,"Hello World");
		}		

3.2	Call	Program MeMySelf;
		var
		int x;
		char y;
		
		module void funcion(int m);
		{
		read(x,y);
		}
		
		main()
		{
		x=3;
		funcion(x);
		write(x,y,"Hello World");
		}
			1
			c
				1
				c
				Hello World

4	Callr	Program MeMySelf;
		var
		int x;
		char y;
		
		module int funcion();
		{
		read(x,y);
		return(3);
		}
		
		main()
		{
		x = funcion();
		write(x,y,"Hello World");
		}
			1
			c
				3
				c
				Hello World

5	Asignación	Program MeMySelf;
		var
		float x;
		char y;
		
		module void funcion(float m);
		{
			1
			c
				3
				1.0
				c
				Hello World

		read(x,y);
		}
		
		main()
		{
		x=3;
		write(x);
		funcion(x);
		write(x,y,"Hello World");
		}		

6	Expresiones	Program MeMySelf;
		var
		float x;
		
		main()
		{
		x=3*2-4/2;
		write(x);
		}
				4.0

7.1	Decisiones	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		if(x==2 & x!=2) then {
		write("True");
		x=3;
		} else {
		write("False");
		x=2;
		}
		write(x);
		}
				False
				2

7.2	Decisiones	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		if(x==2 | x!=2) then {
		write("True");
		x=3;
		} else {
		write("False");
		x=2;
				True
				3

		}
		write(x);
		}		

7.3	Decisiones	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		if(x==2 & x!=2) then {
		write("True");
		x=3;
		}
		write(x);
		}
				1

8.1	Repetición	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		from x = 0 to 3 do {
		write(x);
		}
		}
				0
				1
				2
				3

8.2	Repetición	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		while(x<3) do {
		write(x);
		x=x+1;
		}
		}
				1
				2

8.3	Repetición	Program MeMySelf;
		var
		int x;
		
		main()
		{
		x=1;
		do {
		write(x);
				1
				2

		x=x+1;
		}while(x<3)
		}		

9	Llamadas especiales	Program MeMySelf;
		
		main()
		{
		Color("red");
		Line(100);
		Turn(90);
		Color("blue");
		Point();
		Line(100);
		Turn(90);
		PenUp();
		Line(100);
		Turn(90);
		PenDown();
		Size(10);
		Line(100);
		Turn(90);
		}		

Test Cases que dan error

ID	Test	Código	Consola Input	Error en Consola

2.1.1	Write	Program MeMySelf;
		var
		int x;
		char y;
		
		main()
		{
		write(x,y,"Hello World");
		}
				ERROR: Variable no inicializada

2.2.1	Read	Program MeMySelf;
		
		main()
		{
		read(x,y);
		write(x,y,"Hello World");
		}
				ERROR: Variable no encontrada

3.1.1	Call	Program MeMySelf; var int x; char y; module void funcion();	c	ERROR: Llamada con return a funcion void

		{
		read(x,y);
		}
		
		main()
		{
		x = funcion();
		write(x,y,"Hello World");
		}		

3.2.1	Call	Program MeMySelf;
		var
		int x;
		char y;
		
		module void funcion(int m);
		{
		read(x,y);
		}
		
		main()
		{
		x=3;
		funcion(y);
		write(x,y,"Hello World");
		}
				ERROR: Tipo de parametro esperado invalido

3.2.2	Call	Program MeMySelf;
		var
		int x;
		char y;
		
		module void funcion(int m);
		{
		read(x,y);
		}
		
		main()
		{
		x=3;
		funcion();
		write(x,y,"Hello World");
		}
				ERROR: Cantidad de parametros invalida

4.1	Callr	Program MeMySelf;
		var
		int x;
		char y;
		
		module int funcion();
		{
				ERROR: Llamada a funcion con return en llamada void

		read(x,y);
		return(x);
		}
		
		main()
		{
		funcion();
		write(x,y,"Hello World");
		}		

5.1	Asignación	Program MeMySelf;
		var
		int x;
		char y;
		
		module float funcion();
		{
		read(x,y);
		return(x);
		}
		
		main()
		{
		x = funcion();
		write(x,y,"Hello World");
		}
				ERROR: Tipo de variable invalido en asignacion

5.2	Asignación	Program MeMySelf;
		var
		int x;
		char y;
		
		module void funcion(int m);
		{
		read(x,y);
		}
		
		main()
		{
		x='c';
		write(x);
		funcion(x);
		write(x,y,"Hello World");
		}
				ERROR: Tipo de variable invalido en asignacion

Descripción del proceso general seguido para el desarrollo del proyecto

Se decidió utilizar el lenguaje Python para el compilador, agregando la librería Ply para el léxico y sintaxis, la librería Tortuga para el output gráfico y la librería Copy para simular las variables locales. Las variables del programa se guardaron en un objeto Json, dentro de su correspondiente función y junto con información adicional de cada función.

El proceso seguido fue de cuatro etapas. La primera etapa consistió en utilizar la librería Ply para poder definir el léxico y sintaxis del programa. Se definieron todos los tokens que utilizaría el programa, las palabras reservadas y la estructura del programa. En la segunda etapa se programó la semántica estática, creando el listado de duplas correspondientes y marcando los primeros errores de compilación. En la tercera etapa se programó la semántica dinámica, agregando la ejecución correspondiente a cada dupla, las verificaciones correspondientes y finalmente incluyendo el output gráfico.

Bitácoras

Avance 1

8 de octubre 2020

Entrega

Archivo .py con el léxico y la sintaxis funcionando.

Diferencias entre la GramaticaInicial y GramaticaActual

Las funciones se inicializan con "module <tipo de retorno> nombre_modulo" en lugar de "<tipo de retorno> module nombre_modulo"

Avance 2

14 de octubre 2020

Entrega

Semántica básica

Avance

- Las variables son guardadas por globales y funciones, las variables de funciones no pueden repetirse entre las variables globales
- Los parámetros de funciones son guardados, los parámetros de funciones no pueden repetirse entre las variables globales y las variables de funciones no pueden repetirse entre los parámetros de dicha función
- Las funciones son guardadas con sus variables y parámetros correspondientes

- Las ecuaciones son transformadas en un arreglo de instrucciones en forma de cuádruplos
- Las asignaciones tienen todas sus instrucciones correspondientes

Avance 3

24 de octubre 2020

Entrega

Generación de código intermedio inicial

Avance

- Se agregaron los tokens: '>=', '!=' y '<='
- Las variables del sistema ahora tienen un espacio entre el nombre y su número. Esto ayuda a que el usuario no pueda por accidente crear una variable con el mismo nombre que una variable del sistema, ya que las variables del usuario son sin espacios.
- Se establecieron las estructuras que ira generando el programa en forma de cuádruplos.
- Todos los estatutos mandan su arreglo de instrucciones completo en el formato requerido.
- El arreglo de instrucciones del main es checado en busca de errores de semántica estática.
- El arreglo de instrucciones de cada función es checado en busca de errores de semántica estática.
- Se considero que el usuario puede no declarar variables.

Diferencias entre la GramaticaInicial y GramaticaActual

Se agrego la repetición condicional While Do.

Avance 4

30 de octubre 2020

Entrega

Generación de código intermedio para estatutos no-lineales e inicio de funciones

Avance

- Se reestructuraron las variables virtuales. Se junto en una sola variable la información de todas las funcionas.
- Se creo una tabla de 3 dimensiones con los tipos de variables y tipos de operadores dando como resultado el tipo de valor del resultado.
- Se creo una función dedicada a determinar el tipo de valor, ya sea de un valor constante o de una variable.
- Se creo una función que regresa un apuntador a la variable buscada.

- Se creo una función run para iniciar la ejecución de las instrucciones.
- Se creo una función call para ejecutar las instrucciones de la función activa con variables locales por llamada a la función.
- Se creo una función por cada acción esperada a ejecutar, para todos los estatutos excepto las funciones especiales.
- Se movió el chequeo de la semántica estática para analizar todas las instrucciones en un solo lugar.
- La creación de funciones aparte de main se hizo opcional.
- Se arreglo un problema con las expresiones, el orden de evaluación se tuvo que corregir.
- Todas las constantes del código se guardan en una variable del sistema.
- El estatuto de repetición no condicional, FROM TO, ahora acepta expresiones para la inicialización de la variable y el valor limite.

Reflexión

Fue un proceso interesante, ya que al inicio me encontraba realmente perdida con el lenguaje y la idea. Pensé que sería un logro demasiado difícil para mí. Incluso considere meterme a un equipo solo para poder contar con el apoyo de alguien, pero al ver que la dificultad aumentaba demasiado para mi gusto, decidí hacerlo por mi cuenta y buscar a alguien que me explicara. Cuando comencé con la primera etapa, estuve pegada a la documentación dudando de cada una de mis decisiones. La única persona que encontré que me podía ayudar, había llevado antes de la clase, pero sin permiso para utilizar librerías, asique la idea era que tenía que hacer por mi cuenta la primera parte y luego me explicaría como hacer la semántica. Recuerdo que para cuando terminé la primera parte me encontré de repente programando la semántica dinámica. Realmente no sabía lo que estaba haciendo, pero de alguna forma veía que estaba funcionando. Termine de programar la semántica dinámica casi el mismo día. Para cuando la persona que me iba a ayudar me pregunto en que necesitaba ayuda, ya había terminado el programa. Desde que lo termine me he sentido bastante satisfecha con el resultado, sobre todo por el miedo que le tenía al inicio. También me alegro de no haberme metido en equipo con alguien, más que nada por el hecho de no tener que discutir una decisión con alguien más.

Naomi Macias Honti

Descripción del lenguaje

Nombre del lenguaje

Vintage

Descripción genérica de las principales características del lenguaje

Es un programa orientado a objetos muy parecido a lenguajes como C++ con algunas limitaciones. Puede hacer uso de variables globales y locales, de funciones void y con retorno, de loops como while do, do while y for in do, de decisión e instrucciones de lectura y escritura. Una característica única de este lenguaje es que permite hacer uso de llamadas a funciones especiales que permiten hacer uso de la interfaz gráfica para dibujar en ella.

Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución

Error en Compilación

EC001. Tipo de variable no encontrada: ocurre cuando no se reconoce el tipo de variable o, en caso de ser una variable, no se encuentra definida la variable con un tipo.

EC002. Variable no encontrada: ocurre cuando el nombre de la variable no se encuentra en ningún listado, ni entre las globales, ni entre las locales de la función activa en ese momento.

EC003. Tipo de valores inválidos: ocurre cuando en una expresión se utilizan variables diferentes del tipo int o float.

EC004. PROGRAM01: error en código del programa, consultar a un programador del lenguaje.

EC005. Tipo de variable invalido en asignación: ocurre cuando se intenta ingresar un valor incompatible en una variable. Por ejemplo, que se intente guardar un valor no numérico en una variable numérica.

EC006. Return en función void: ocurre cuando se encuentra un return dentro de una función declarada como void.

EC007. Tipo de return invalido: ocurre cuando el tipo de valor en un return no coincide con el tipo de función donde se encuentra el return.

EC008. Llamada a función invalida: ocurre cuando se hace una llamada a una función y dicha función no fue declarada en el programa.

EC009. Llamada con return a función void: ocurre cuando se hace una llamada a una función esperando un valor de regreso y dicha función es de tipo void.

EC010. Cantidad de parámetros invalida: ocurre cuando se hace una llamada a una función y la cantidad de parámetros enviados en la llamada es diferente a la cantidad de parámetros declarados en la función.

EC011. Tipo de parámetro esperado invalido en return: ocurre cuando se intenta hacer una llamada a una función y alguno de los tipos de los parámetros es diferente a su tipo de parámetro esperado según la definición de la función y el orden de los parámetros enviados.

EC012. Llamada a función con return en llamada void: ocurre cuando se hace una llamada a una función que no es void y no se utiliza el valor de retorno.

EC013. Llamada a main: ocurre cuando se hace una llamada a la función main.
EC014. Tipo de parámetro invalido: ocurre cuando se intenta hacer una llamada a una función y alguno de los tipos de los parámetros es diferente a su tipo de parámetro esperado según la definición de la función y el orden de los parámetros enviados.
EC015. PROGRAM02: error en código del programa, consultar a un programador del lenguaje.
EC016. Nombre de variable repetido, variable global declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre las variables globales.
EC017. Nombre de variable repetido, parámetro de función declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre los parámetros de la función.
EC018. Nombre de variable repetido, variable de función declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre las variables de la función.
EC019. Nombre de función invalido: ocurre cuando se intenta declarar una función con el nombre de otra función previamente declarada.
EC020. Nombre de parámetro invalido: ocurre cuando el nombre del parámetro que se intenta declarar, ya se encuentra declarada entre las variables globales o entre los parámetros previamente declarados.

Error en Ejecución

EE001. OP01: error en código del programa, consultar a un programador del lenguaje.
EE002. Variable no inicializada: ocurre cuando se quiere utilizar una variable en una expresión, llamada o escritura y la variable no cuenta con un valor en ese momento.
EE003. No llego a un return la función: ocurre cuando una función, de tipo no void, no regresa algún valor.
EE004. CALL01: error en código del programa, consultar a un programador del lenguaje.
EE005. CALLF01: error en código del programa, consultar a un programador del lenguaje.
EE006. Longitud de carácter mayor a 1: ocurre cuando en una instrucción read se recibe del usuario un string mayor a un carácter y el valor esperado es un char.
EE007. READ01: error en código del programa, consultar a un programador del lenguaje.

Descripción del compilador

Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto

Equipo utilizado: Windows 7.

Lenguaje: Python.

Librerías: Ply, Copy y Tortuga.

Programas: Visual Studio Code para programar, compilar y ejecutar el código y Github para administrar las versiones y cambios.

Descripción del Análisis de Léxico

Tokens

T001. AND = ‘&’
T002. OR = ‘|’
T003. LE = ‘<=’
T004. LT = ‘<’
T005. GE = ‘>=’
T006. GT = ‘>’
T007. EQ = ‘==’
T008. NE = ‘!=’
T009. EQUAL = ‘=’
T010. LPAREN = ‘(‘
T011. RPAREN = ‘)’
T012. LPAREN2 = ‘{‘
T013. LPAREN2 = ‘}’
T014. PUNCOM = ‘;’
T015. COMA = ‘,’
T016. NINT = '\d+'
T017. NFLOAT = '\d+\.\d+'
T018. SCHAR = '\'.\''
T019. SSTRING = '\".*\"'
T020. SVAR = '[a-zA-Z][a-zA-Z0-9]*'

Palabras Reservadas

PR001. PROGRAM = Program
PR002. MAIN = main
PR003. VAR = var
PR004. INT = int
PR005. FLOAT = float
PR006. CHAR = char

PR007. VOID = void
PR008. MODULE = module
PR009. RETURN = return
PR010. READ = read
PR011. WRITE = write
PR012. IF = if
PR013. THEN = then
PR014. ELSE = else
PR015. DO = do
PR016. WHILE = while
PR017. TRUE = true
PR018. FALSE = false
PR019. FROM = from
PR020. TO = to
PR021. TURN = Turn
PR022. LINE = Line
PR023. POINT = Point
PR024. PENUP = PenUp
PR025. PENDOWN = PenDown
PR026. COLOR = Color
PR027. SIZE = Size

Descripción del Análisis de Sintaxis

program : programInicio decvar decfuntemp programMain

decfuntemp : decfunCodigo decfuntemp
| empty

programMain : programMainIni LPAREN2 estatutos RPAREN2

programMainIni : MAIN LPAREN RPAREN

programInicio : PROGRAM SVAR PUNCOM

decvar : VAR decvar2
| empty

decvar2 : tipo SVAR nomvar decvar2
| empty

nomvar : COMA SVAR nomvar
| PUNCOM

tipo : INT
| FLOAT
| CHAR

decfunCodigo : decfunCodigoIni LPAREN funpara RPAREN PUNCOM decvar LPAREN2 estatutos RPAREN2

decfunCodigoIni : MODULE funtipo SVAR

funtipo : tipo
| VOID

funpara : tipo SVAR funpara2
| empty

funpara2 : COMA tipo SVAR funpara2
| empty

estatutos : asignacion estatutos
| llamada estatutos
| lectura estatutos

| escritura estatutos

| desicion estatutos

| repeticion estatutos

| funespecial estatutos

| retorno

| empty

asignacion : SVAR EQUAL asitipos PUNCOM

asitipos : expr

| SCHAR

expr : exprCode

exprCode : expr1 exprCodeT

exprCodeT : '+' expr1 exprCodeT

| '-' expr1 exprCodeT

| empty

expr1 : expr2 expr1T

expr1T : '*' expr2 expr1T

| '/' expr2 expr1T

| empty

expr2 : LPAREN exprCode RPAREN

| term

term : NINT

| NFLOAT

| SVAR posfun

posfun : LPAREN parametros RPAREN

| empty

parametros : asitipos parametros2

| empty

parametros2 : COMA asitipos parametros2

| empty

llamada : SVAR LPAREN parametros RPAREN PUNCOM

retorno : RETURN LPAREN asitipos RPAREN PUNCOM

lectura : READ LPAREN SVAR lectura2 RPAREN PUNCOM

lectura2 : COMA SVAR lectura2

| empty

escritura : WRITE LPAREN escritura2 escritura3 RPAREN PUNCOM

escritura2 : SSTRING

| asitipos

escritura3 : COMA escritura2 escritura3

| empty

desicion : IF LPAREN expresion RPAREN THEN LPAREN2 estatutos RPAREN2 desicion2

desicion2 : ELSE LPAREN2 estatutos RPAREN2

| empty

repeticion : condicional

| nocondicional

condicional : DO LPAREN2 estatutos RPAREN2 WHILE LPAREN expresion RPAREN

| WHILE LPAREN expresion RPAREN DO LPAREN2 estatutos RPAREN2

expresion : comp1 expresion2

expresion2 : comp2 comp1 expresion2

| empty

comp2 : AND

| OR

comp1 : asitipos comp3 asitipos

| TRUE

| FALSE

comp3 : EQ

| GT

| GE

| LT

| LE

| NE

nocondicional : FROM SVAR EQUAL expr TO expr DO LPAREN2 estatutos RPAREN2

funespecial : LINE LPAREN expr RPAREN PUNCOM

| TURN LPAREN expr RPAREN PUNCOM

| SIZE LPAREN expr RPAREN PUNCOM

| POINT LPAREN RPAREN PUNCOM

| PENUP LPAREN RPAREN PUNCOM

| PENDOWN LPAREN RPAREN PUNCOM

| COLOR LPAREN SSTRING RPAREN PUNCOM

Descripción de Generación de Código Intermedio y Análisis Semántico

S001. [program] = Ultima instrucción de compilación. Repasa cada función declarada y analiza los cuádruplos generados de la función. Busca errores de semántica dinámica. Al finalizar llama la función Run para ejecutar el código.

S002. [decfuntemp] = Sin código.

S003. [programMain] = Guarda el arreglo de duplas recibido de *estatutos* en "run" dentro de "main".

S004. [programMainIni] = Coloca como función activa la función "main".

S005. [programInicio] = Coloca como función activa la función "main".

S006. [decvar] = Sin código.

S007. [decvar2] = En caso de no ser *empty*, crea un arreglo con los valores recibidos en *SVAR* y *nomvar*. Repasa el arreglo, comprueba que el nombre de la variable no se encuentre ya declarado y agrega la variable en la tabla de variables de la función activa con el tipo de variable recibido de *tipo*.

S008. [nomvar] = En caso de no ser *PUNCOM*, va creando un arreglo con los valores recibidos en *SVAR*.

S009. [tipo] = Regresa el valor encontrado.

S010. [decfunCodigo] = Guarda el arreglo de duplas recibido de estatutos en "run" dentro de la función activa.

S011. [decfunCodigoIni] = Coloca como función activa el nombre de función recibido en *SVAR*. Crea la tabla de información de la función, indicando el nombre, tipo de función (recibido de *funtipo*) y el valor inicial de sus variables temporales.

S012. [funtipo] = Regresa el valor encontrado.

S013. [funpara] = En caso de no ser *empty*, crea un arreglo de duplas con el nombre de la variable y el tipo de variable, luego recorre el arreglo comprobando que los nombres no hayan sido declarados anteriormente y, en caso de no ser así, agregando la variable a la tabla de la función activa junto con su tipo.

S014. [funpara2] = En caso de no ser *empty*, regresa un arreglo de duplas con el nombre de la variable y el tipo de variable.
S015. [estatutos] = En caso de no ser *empty*, regresa un arreglo con los arreglos recibidos de los diferentes estatutos.
S016. [asignacion] = Envía los cuádruplos necesarios para llegar al resultado, si hay alguno, y agrega un cuádruplo asignando el valor a la variable recibida en *SVAR*.
S017. [asitipos] = Envía la lista de cuádruplos recibidos en *expr* o una variable temporal con el valor recibido en *SCHAR*.
S018. [expr] = Recibe un arreglo de vector polaco y lo convierte en un listado de cuádruplos.
S019. [exprCode] = Envía un arreglo de vector polaco juntando los arreglos que recibe de *expr1* y *exprCodeT*.
S020. [exprCodeT] = En caso de no ser empty, envía un arreglo de vector polaco juntando los arreglos que recibe de *expr1* y *exprCodeT* y agrega el valor de signo recibido.
S021. [expr1] = Envía un arreglo de vector polaco juntando los arreglos que recibe de *expr2* y *expr1T*.
S022. [expr1T] = En caso de no ser empty, envía un arreglo de vector polaco juntando los arreglos que recibe de *expr2* y *expr1T* y agrega el valor de signo recibido.
S023. [expr2] = Envía un arreglo vector polaco recibido de *exprCode* o el valor recibido de *term* en un arreglo.
S024. [term] = Regresa el valor encontrado, si recibe un valor de *posfun*, regresa la función con su nombre y parámetros.
S025. [posfun] = En caso de no ser *empty*, envía un arreglo con los parámetros recibidos de *parametros*.
S026. [parametros] = En caso de no ser *empty*, regresa un arreglo juntando lo recibido de *asitipos* y *parametros2*.
S027. [parametros2] = En caso de no ser *empty*, regresa un arreglo juntando lo recibido de *asitipos* y *parametros2*.
S028. [llamada] = Envía los cuádruplos necesarios para llegar a los resultados, si hay algunos, y agrega un cuádruplo llamando a la función recibida en *SVAR* con los parámetros recibidos de *parametros*.
S029. [retorno] = Envía los cuádruplos necesarios para llegar al resultado, si hay alguno, y agrega un cuádruplo de return junto con el ultimo valor de *asitipos*.
S030. [lectura] = Agrega un cuádruplo de read por cada valor recibido de *lectura2*.
S031. [lectura2] = En caso de no ser *empty*, egresa el arreglo de los valores recibidos en *SVAR* y *lectura2*.
S032. [escritura] = Agrega un cuádruplo de *write* por cada valor recibido de *escritura2* y *escritura3*.
S033. [escritura2] = Regresa la variable recibida *SVAR* o una variable temporal con el valor recibido de *SSTRING*.
S034. [escritura3] = En caso de no ser *empty*, envía un arreglo juntando lo recibido de *escritura2* y *escritura3*.
S035. [desicion] = Envía un arreglo juntando el arreglo recibido de *expresion*, un cuádruplo creado de gotof, el arreglo recibido de *estatutos* y, en caso de que se reciba algo de *desicion2*, un cuádruplo creado de goto y el arreglo recibido de *desicion2*.

S036. [desicion2] = En caso de no ser *empty*, regresa el arreglo recibido de *estatutos*.
S037. [repeticion] = Regresa el valor encontrado.
S038. [condicional] = Envía un arreglo juntando los arreglos recibidos de *expresion* y *estatutos* y dos cuádruplos creados de goto y gotof, cada uno en su correspondiente lugar.
S039. [expresion] = Regresa un arreglo juntando el arreglo recibido de *comp1* con el arreglo de cuádruplos creados para llegar al resultado siguiendo el arreglo de duplas recibido de *expresion2*.
S040. [expresion2] = Envía un arreglo con la dupla creada con los valores encontrados en *comp2* y *comp1* y los valores recibidos de *expresion2*.
S041. [comp1] = Envía un arreglo juntando los arreglos recibidos de *asitipos* y un cuádruplo creado de comparación.
S042. [comp2] = Regresa el valor encontrado.
S043. [comp3] = Regresa el valor encontrado.
S044. [nocondicional] = Envía un arreglo juntando los recibidos de *expr* y *estatutos* y los cuádruplos creados de asignación, suma, gotof y goto necesarios en sus lugares correspondientes.
S045. [funespecial] = Envía un arreglo juntando el recibido de *expr*, si hay alguno, y el cuádruplo creado con la llamada a la función especial.

Tabla de consideraciones semánticas

Tipo1	Tipo2	Operador	Resultado
Int	Int	+	Int
Int	Int	-	Int
Int	Int	*	Int
Int	Int	/	Float
Int	Int	==	Bool
Int	Int	!=	Bool
Int	Int	<=	Bool
Int	Int	<	Bool
Int	Int	>=	Bool
Int	Int	>	Bool
Int	Int	&	Error
Int	Int	|	Error
Int	Float	+	Float
Int	Float	-	Float
Int	Float	*	Float
Int	Float	/	Float
Int	Float	==	Bool
Int	Float	!=	Bool
Int	Float	<=	Bool
Int	Float	<	Bool
Int	Float	>=	Bool
Int	Float	>	Bool
Int	Float	&	Error

Int	Float	|	Error
Int	Char	+	Error
Int	Char	-	Error
Int	Char	*	Error
Int	Char	/	Error
Int	Char	==	Bool
Int	Char	!=	Bool
Int	Char	<=	Error
Int	Char	<	Error
Int	Char	>=	Error
Int	Char	>	Error
Int	Char	&	Error
Int	Char	|	Error
Int	Bool	+	Error
Int	Bool	-	Error
Int	Bool	*	Error
Int	Bool	/	Error
Int	Bool	==	Bool
Int	Bool	!=	Bool
Int	Bool	<=	Error
Int	Bool	<	Error
Int	Bool	>=	Error
Int	Bool	>	Error
Int	Bool	&	Error
Int	Bool	|	Error
Float	Int	+	Float
Float	Int	-	Float
Float	Int	*	Float
Float	Int	/	Float
Float	Int	==	Bool
Float	Int	!=	Bool
Float	Int	<=	Bool
Float	Int	<	Bool
Float	Int	>=	Bool
Float	Int	>	Bool
Float	Int	&	Error
Float	Int	|	Error
Float	Float	+	Float
Float	Float	-	Float
Float	Float	*	Float
Float	Float	/	Float
Float	Float	==	Bool
Float	Float	!=	Bool
Float	Float	<=	Bool
Float	Float	<	Bool
Float	Float	>=	Bool

Float	Float	>	Bool
Float	Float	&	Error
Float	Float	|	Error
Float	Char	+	Error
Float	Char	-	Error
Float	Char	*	Error
Float	Char	/	Error
Float	Char	==	Bool
Float	Char	!=	Bool
Float	Char	<=	Error
Float	Char	<	Error
Float	Char	>=	Error
Float	Char	>	Error
Float	Char	&	Error
Float	Char	|	Error
Float	Bool	+	Error
Float	Bool	-	Error
Float	Bool	*	Error
Float	Bool	/	Error
Float	Bool	==	Bool
Float	Bool	!=	Bool
Float	Bool	<=	Error
Float	Bool	<	Error
Float	Bool	>=	Error
Float	Bool	>	Error
Float	Bool	&	Error
Float	Bool	|	Error
Float	Int	+	Int
Char	Int	-	Int
Char	Int	*	Int
Char	Int	/	Float
Char	Int	==	Bool
Char	Int	!=	Bool
Char	Int	<=	Bool
Char	Int	<	Bool
Char	Int	>=	Bool
Char	Int	>	Bool
Char	Int	&	Error
Char	Int	|	Error
Char	Float	+	Error
Char	Float	-	Error
Char	Float	*	Error
Char	Float	/	Error
Char	Float	==	Bool
Char	Float	!=	Bool
Char	Float	<=	Error

Char	Float	<	Error
Char	Float	>=	Error
Char	Float	>	Error
Char	Float	&	Error
Char	Float	|	Error
Char	Char	+	Error
Char	Char	-	Error
Char	Char	*	Error
Char	Char	/	Error
Char	Char	==	Bool
Char	Char	!=	Bool
Char	Char	<=	Error
Char	Char	<	Error
Char	Char	>=	Error
Char	Char	>	Error
Char	Char	&	Error
Char	Char	|	Error
Char	Bool	+	Error
Char	Bool	-	Error
Char	Bool	*	Error
Char	Bool	/	Error
Char	Bool	==	Bool
Char	Bool	!=	Bool
Char	Bool	<=	Error
Char	Bool	<	Error
Char	Bool	>=	Error
Char	Bool	>	Error
Char	Bool	&	Error
Char	Bool	|	Error
Bool	Int	+	Error
Bool	Int	-	Error
Bool	Int	*	Error
Bool	Int	/	Error
Bool	Int	==	Bool
Bool	Int	!=	Bool
Bool	Int	<=	Error
Bool	Int	<	Error
Bool	Int	>=	Error
Bool	Int	>	Error
Bool	Int	&	Error
Bool	Int	|	Error
Bool	Float	+	Error
Bool	Float	-	Error
Bool	Float	*	Error
Bool	Float	/	Error
Bool	Float	==	Bool

Bool	Float	!=	Bool
Bool	Float	<=	Error
Bool	Float	<	Error
Bool	Float	>=	Error
Bool	Float	>	Error
Bool	Float	&	Error
Bool	Float	|	Error
Bool	Char	+	Error
Bool	Char	-	Error
Bool	Char	*	Error
Bool	Char	/	Error
Bool	Char	==	Bool
Bool	Char	!=	Bool
Bool	Char	<=	Error
Bool	Char	<	Error
Bool	Char	>=	Error
Bool	Char	>	Error
Bool	Char	&	Error
Bool	Char	|	Error
Bool	Bool	+	Error
Bool	Bool	-	Error
Bool	Bool	*	Error
Bool	Bool	/	Error
Bool	Bool	==	Bool
Bool	Bool	!=	Bool
Bool	Bool	<=	Error
Bool	Bool	<	Error
Bool	Bool	>=	Error
Bool	Bool	>	Error
Bool	Bool	&	Bool
Bool	Bool	|	Bool

Descripción detallada del proceso de Administración de Memoria usado en la compilación

Tabla de datos

```
# Estructura para variables temporales dentro de var
# {
# "nombre_de_funcion ##": {
#    "type": "tipo de variable",
#    "value": "valor de la variable"
#    },
# "contador sys": ##
# }

# Estructura
```

```
# {
# "nombre_de_funcion" : {
#    "type": "tipo de funcion"
#    "var": {
#         "nombre_de_variable":{...}
#    },
#    "param": {
#         "nombre_de_parametro":{...}
#    },
#    "run": [(...)]
#    }
# "active sys": "nombre de funcion"
# }
# ejemplo de llamada
# variables["nombre_de_funcion"]["var"]["nombre_de_variable"]["type"]
# inicializado con la funcion main
variables = {"main":{"type":"void","param":{},"run":[],"var":{"contador sys"
: 0}}}
```

Toda la memoria se guarda en la variable global llamada “variables”. En el primer nivel se guardan el nombre de las funciones, una variable llamada “active sys” utilizada para poner el nombre de la función que se está analizando en el momento (la función activa) y una variable llamada “output avtive” utilizada para señalar si el usuario utiliza el output gráfico. Dentro de cada función se guardan cuatro datos: “type” que señala el tipo de función; “var” que contiene todas las variables declaradas dentro de la función, así como sus variables temporales y una variable especial llamada “contador sys” que contiene el número de variables temporales definidas en el momento; “param” que contiene todos los parámetros definidos con la función; “run” que contiene el arreglo de cuádruplos con las instrucciones de la función. Para crear una nueva variable temporal se combina el nombre de la función y el siguiente número del contador. Cada variable y parámetro es guardada con los valores de “type” que señala el tipo de la variable y “value” que contiene el valor de la variable.

Cuádruplos

```
# Estructura
# (op, opdo1, opdo2, result)
# (=, var, null, result)
# (callr, func, param, result)
# (callf, func, param, null)
# (call, func, param, null)
# (goto, null, null, pos)
# (gotof, value, null, pos)
# (return, null, null, result)
# (write, null, null, result)
```

```
# (read, null, null, result)
```

(op, opdo1, opdo2, result): se utiliza el signo enviado en *op* para evaluar *opdo1* y *opdo2* y guarda el resultado en *result*.

(=, var, null, result): guarda el valor de *var* en *result*.

(callr, func, param, result): llama a la función *func* con los parámetros *param* y guarda el valor de retorno esperado en *result*.

(call, func, param, null): llama a la función *func* con los parámetros *param*.

(callf, func, param, null): llama a la función especial *func* con los parámetros *param*.

(goto, null, null, pos): le suma al contador actual el valor de *pos*.

(gotof, value, null, pos): comprueba si el valor en *value* es falso y, en caso de ser asi, le suma al contador actual el valor de *pos*.

(return, null, null, result): regresa el valor de *result*.

(write, null, null, result): escribe en consola el valor de *result*.

(read, null, null, result): recibe del usuario un valor que guarda en *result*.

Funciones especiales

```
# Funciones especiales
# PenUp() => turtle.penup()
# PenDown() => turtle.pendown()
# Point() => turle.dot(10, 0, 0, 0)
# Turn(nomber) => turtle.left(nomber)
# Line(nomber) => turtle.forward(nomber)
# Color(string) => turtle.pencolor(string)
# Size(nomber) => turtle.pensize(nomber)
```

PenUp: deja de pintar el camino que sigue la tortuga.

PenDown: dibuja el camino que sigue la tortuga.

Point: coloca un punto en la posición actual de la tortuga.

Turn(nomber): gira el valor de *nomber* en grados a la tortuga hacia la izquierda.

Line(nomber): hace avanzar a la tortuga el valor de *nomber* hacia delante.

Color(string): define el color de las líneas como el valor de *string*.

Size(nomber): define el tamaño de las líneas como el valor de *nomber*.

Descripción de la máquina virtual

Equipo de cómputo, lenguaje y utilerías especiales usadas

Se utilizo las mismas especificadas en el compilador.

Descripción detallada del proceso de Administración de Memoria en ejecución

Se utilizo la estructura señalada en la Tabla de datos. Para manejar la diferencia entre variables globales y locales, cuando se corre la función main, se utiliza un apuntador a las variables de main dentro de "variables", cuando se llama a una función, sus parámetros y variables se les hace un "deepcopy" que, en lugar de mandar un apuntador a las variables originales, las copia en otra parte de la memoria y manda el apuntador a las copias.

Ejemplo de llamada al main

```
call("main",variables["main"]["param"],variables["main"]["var"])
```

Ejemplo de llamada a función

```
call(funcion,TempParametros,copy.deepcopy(variables[funcion]["var"]))
```

En este caso, los parámetros se definen antes como:

```
TempParam = copy.deepcopy(variables[funcion]["param"])
```

y se les asigna el valor correspondiente antes de ser enviados en la llamada

Pruebas del funcionamiento del lenguaje

ID	Código	Cuádruplos	Input	Output

1	Program MeMySelf;
	
	main()
	{
	write("Hello World");
	}
		*** main
		1 : write | None | None | main 1
				Hello World

2	Program MeMySelf;
	var
	int x;
	char y;
	
	main()
	{
	read(x,y);
	write(x,y,"Hello World",4+1,x*3);
	}
		*** main
		1 : read | None | None | x
		2 : read | None | None | y
		3 : write | None | None | x
		4 : write | None | None | y
		5 : write | None | None | main 1
		6 : + | main 2 | main 3 | main 4
		7 : write | None | None | main 4
		8 : * | x | main 5 | main 6
		9 : write | None | None | main 6
			2
			c
				2
				c
				Hello World
				5
				6

3.1	Program MeMySelf;
	var
	int x;
	char y;
	
	module void funcion();
	{
	read(x,y);
	}
	
	main()
	{
	funcion();
	write(x,y,"Hello World");
	}
		*** main
		1 : call | funcion | [] | None
		2 : write | None | None | x
		3 : write | None | None | y
		4 : write | None | None | main 1
		*** funcion
		1 : read | None | None | x
		2 : read | None | None | y
			1
			c
				1
				c
				Hello World

3.2	Program MeMySelf;
	var
	int x;
	char y;
	
	module void funcion(int m);
	{
	read(x,y);
	}
	
	main()
	{
	x=3;
		*** main
		1 : = | main 1 | None | x
		2 : call | funcion | ['x'] | None
		3 : write | None | None | x
		4 : write | None | None | y
		5 : write | None | None | main 2
		*** funcion
		1 : read | None | None | x
		2 : read | None | None | y
			1
			c
				1
				c
				Hello World

	funcion(x);
	write(x,y,"Hello World");
	}			

4	Program MeMySelf;
	var
	int x;
	char y;
	
	module int funcion();
	{
	read(x,y);
	return(3);
	}
	
	main()
	{
	x = funcion();
	write(x,y,"Hello World");
	}
		*** main
		1 : callr | funcion | [] | main 1
		2 : = | main 1 | None | x
		3 : write | None | None | x
		4 : write | None | None | y
		5 : write | None | None | main 2
		*** funcion
		1 : read | None | None | x
		2 : read | None | None | y
		3 : return | None | None | funcion 1
			1
			c
				3
				c
				Hello World

5	Program MeMySelf;
	var
	float x;
	char y;
	
	module void funcion(float m);
	{
	read(x,y);
	}
	
	main()
	{
	x=3;
	write(x);
	funcion(x);
	write(x,y,"Hello World");
	}
		*** main
		1 : = | main 1 | None | x
		2 : write | None | None | x
		3 : call | funcion | ['x'] | None
		4 : write | None | None | x
		5 : write | None | None | y
		6 : write | None | None | main 2
		*** funcion
		1 : read | None | None | x
		2 : read | None | None | y
			1
			c
				3
				1.0
				c
				Hello World

6	Program MeMySelf;
	var
	float x;
	
	main()
	{
	x=3*2-4/2;
	write(x);
	}
		*** main
		1 : * | main 1 | main 2 | main 5
		2 : / | main 3 | main 4 | main 6
		3 : - | main 5 | main 6 | main 7
		4 : = | main 7 | None | x
		5 : write | None | None | x
				4.0

7.1	Program MeMySelf; var	*** main 1 : = | main 1 | None | x		False 2

	int x;
	
	main()
	{
	x=1;
	if(x==2 & x!=2) then {
	write("True");
	x=3;
	} else {
	write("False");
	x=2;
	}
	write(x);
	}
		2 : == | x | main 2 | main 3
		3 : != | x | main 4 | main 5
		4 : & | main 3 | main 5 | main 6
		5 : gotof | main 6 | None | 4
		6 : write | None | None | main 7
		7 : = | main 8 | None | x
		8 : goto | None | None | 3
		9 : write | None | None | main 9
		10 : = | main 10 | None | x
		11 : write | None | None | x		

7.2	Program MeMySelf;
	var
	int x;
	
	main()
	{
	x=1;
	if(x==2 | x!=2) then {
	write("True");
	x=3;
	} else {
	write("False");
	x=2;
	}
	write(x);
	}
		*** main
		1 : = | main 1 | None | x
		2 : == | x | main 2 | main 3
		3 : != | x | main 4 | main 5
		4 : | | main 3 | main 5 | main 6
		5 : gotof | main 6 | None | 4
		6 : write | None | None | main 7
		7 : = | main 8 | None | x
		8 : goto | None | None | 3
		9 : write | None | None | main 9
		10 : = | main 10 | None | x
		11 : write | None | None | x
				True
				3

7.3	Program MeMySelf;
	var
	int x;
	
	main()
	{
	x=1;
	if(x==2 & x!=2) then {
	write("True");
	x=3;
	}
	write(x);
	}
		*** main
		1 : = | main 1 | None | x
		2 : == | x | main 2 | main 3
		3 : != | x | main 4 | main 5
		4 : & | main 3 | main 5 | main 6
		5 : gotof | main 6 | None | 3
		6 : write | None | None | main 7
		7 : = | main 8 | None | x
		8 : write | None | None | x
				1

8.1	Program MeMySelf; var int x; main()	*** main 1 : = | main 1 | None | x 2 : = | main 2 | None | x 3 : <= | x | main 3 | main 4 4 : gotof | main 4 | None | 5		0 1 2 3

	{
	x=1;
	from x = 0 to 3 do {
	write(x);
	}
	}
		5 : write | None | None | x
		6 : + | x | main 6 | main 5
		7 : = | main 5 | None | x
		8 : goto | None | None | -5		

8.2	Program MeMySelf;
	var
	int x;
	
	main()
	{
	x=1;
	while(x<3) do {
	write(x);
	x=x+1;
	}
	}
		*** main
		1 : = | main 1 | None | x
		2 : < | x | main 2 | main 3
		3 : gotof | main 3 | None | 5
		4 : write | None | None | x
		5 : + | x | main 4 | main 5
		6 : = | main 5 | None | x
		7 : goto | None | None | -5
				1
				2

8.3	Program MeMySelf;
	var
	int x;
	
	main()
	{
	x=1;
	do {
	write(x);
	x=x+1;
	}while(x<3)
	}
		*** main
		1 : = | main 1 | None | x
		2 : write | None | None | x
		3 : + | x | main 2 | main 3
		4 : = | main 3 | None | x
		5 : < | x | main 4 | main 5
		6 : gotof | main 5 | None | 2
		7 : goto | None | None | -5
				1
				2

9	Program MeMySelf; main() { Color("red"); Line(100); Turn(90); Color("blue"); Point(); Line(100); Turn(90); PenUp(); Line(100); Turn(90); PenDown(); Size(10); Line(100); Turn(90);	*** main 1 : callf | Color | ['main 1'] | None 2 : callf | Line | ['main 2'] | None 3 : callf | Turn | ['main 3'] | None 4 : callf | Color | ['main 4'] | None 5 : callf | Point | [] | None 6 : callf | Line | ['main 5'] | None 7 : callf | Turn | ['main 6'] | None 8 : callf | PenUp | [] | None 9 : callf | Line | ['main 7'] | None 10 : callf | Turn | ['main 8'] | None 11 : callf | PenDown | [] | None 12 : callf | Size | ['main 9'] | None 13 : callf | Line | ['main 10'] | None 14 : callf | Turn | ['main 11'] | None		

	}			

Documentación del código del proyecto

Run

Función

Comienza la ejecución de las instrucciones empezando por la función inicial main.

Código

```
def run():
     # Declara a main como función activa
     variables["active sys"] = "main"
     # Declara False al output grafico
     variables["output active"] = False
     # Comienza la ejecución con un apuntador a las variables y parámetros g
lobales de main
     call("main",variables["main"]["param"],variables["main"]["var"])
     # Manda a llamar a la última instrucción del output grafico
     closeOutput()
```

Read

Función

Se encarga de pedirle un dato al usuario y se asegura de que el dato ingresado sea del *tipo* esperado.

Código

```
def read(tipo):
     if tipo == "int":
          return int(input())
     elif tipo == "float":
          return float(input())
     elif tipo == "char":
          inputTemp = str(input())
          if len(inputTemp) != 1:
               print("ERROR: Longitud de caracter mayor a 1")
               raise CalcError("Input invalido")
          return inputTemp
     print("ERROR: READ01")
     raise CalcError("Error en sistema")
```

Parámetros

tipo: tipo de dato esperado a recibir por el usuario.

Retorno

Regresa el dato recibido por el usuario.

CloseOutput

Función

En caso de que el output grafico haya sido utilizado, corre la última instrucción de cierre: *exitonclick*, evita que el output grafico se cierre automáticamente en cuanto termine de ejecutar las instrucciones.

Código

```
def closeOutput():
     # Comprueba que el output grafico haya sido abierto
     if variables["output active"] == True:
          turtle.exitonclick()
```

Callf

Función

Ejecuta la función necesaria para manipular el output gráfico.

Código

```
def callf(func,param):
     # Llama a abrir el output grafico
     openOutput()
     if func == "PenUp":
          turtle.penup()
     elif func == "PenDown":
          turtle.pendown()
     elif func == "Point":
          turtle.dot(10,0,0,0)
     elif func == "Turn":
          turtle.left(param[0])
     elif func == "Line":
          turtle.forward(param[0])
     elif func == "Color":
          turtle.pencolor(param[0])
     elif func == "Size":
          turtle.pensize(param[0])
     else:
```

```
            print("ERROR: CALLF01")
            raise CalcError("Error en sistema")
```

Parámetros

func: contiene el nombre de la función que se quiere ejecutar.

param: contiene los valores de los parámetros necesarios para la función de ser necesario.

OpenOutput

Función

Ejecuta la primera instrucción necesaria para abrir el output gráfico. Utiliza un booleano global para asegurarse de solo ejecutar la instrucción una vez.

Código

```
def openOutput():
     # Comprueba que el output grafico no haya sido abierto ya
     if variables["output active"] == False:
          variables["output active"] = True
          turtle.shape("turtle")
```

Call

Función

Utiliza un contador para ejecutar en orden cada instrucción de la función *function*, por cada instrucción encontrada manda a llamar la correspondiente instrucción o función necesaria.

Código

```
def call(function,param,var):
     contador = 0
     while contador < len(variables[function]["run"]):
          a = variables[function]["run"][contador][0]
          b = variables[function]["run"][contador][1]
          c = variables[function]["run"][contador][2]
          d = variables[function]["run"][contador][3]
          if a == "read":
               # Recibe el apuntador a la variable
               vard = buscaVariable(param,var,d)
               vard["value"] = read(vard["type"])
          elif (a == '*' or a == '/' or a == '-
' or a == '+' or a == '==' or a == '>' or a == '<' or a == '<=' or a == '>='
 or a == '!=' or a == '&' or a == '|'):
```

```
                # Recibe el apuntador a la variable
                varb = buscaVariable(param,var,b)
                # Comprueba que la variable se encuentre inicializada
                if varb.get("value") == None:
                     print("ERROR: Variable no inicializada")
                     raise CalcError("Expresion invalida")
                varc = buscaVariable(param,var,c)
                if varc.get("value") == None:
                     print("ERROR: Variable no inicializada")
                     raise CalcError("Expresion invalida")
                vard = buscaVariable(param,var,d)
                # Recibe el resultado de la operacion
                vard["value"] = op(a,varb["value"],varc["value"])
           elif a == '=':
                # Recibe el apuntador a la variable
                varb = buscaVariable(param,var,b)
                # Comprueba que la variable se encuentre inicializada
                if varb.get("value") == None:
                     print("ERROR: Variable no inicializada")
                     raise CalcError("Expresion invalida")
                vard = buscaVariable(param,var,d)
                # Ejecuta la asignacion
                vard["value"] = varb["value"]
           elif a == "callr" or a == "call":
                # Activa la nueva funcion a ejecutar
                variables["active sys"] = b
                # Hace una copia local de los parametros
                tparam = copy.deepcopy(variables[b]["param"])
                tcont = 0
                # Inicializa cada parametro con el valor recibido
                for x in tparam:
                     varc = buscaVariable(param,var,c[tcont])
                     if varc.get("value") == None:
                          print("ERROR: Variable no inicializada")
                          raise CalcError("Expresion invalida")
                     tparam[x]["value"] = varc["value"]
                     tcont = tcont + 1
                if a == "callr":
                     # Ejecuta la funcion con parametros locales, ya iniciali
zados, y envia una copia local de las variables
                     # Guarda el retorno de la funcion en una variable tempor
al
                     temp = call(b,tparam,copy.deepcopy(variables[b]["var"]))
                     # Activa la funcion anterior
                     variables["active sys"] = function
```

```
                    # Si la funcion regresa un valor, lo guarda en la variab
le asignada
                    if temp == "Sys None":
                         print("ERROR: No llego a un return la funcion")
                         raise CalcError("Estatuto faltante")
                    else:
                         vard = buscaVariable(param,var,d)
                         vard["value"] = temp
               else:
                    # Ejecuta la funcion con parametros locales, ya iniciali
zados, y envia una copia local de las variables
                    call(b,tparam,copy.deepcopy(variables[b]["var"]))
                    # Activa la funcion anterior
                    variables["active sys"] = function
          elif a == "gotof":
               varb = buscaVariable(param,var,b)
               # Si el valor de la variable es falso, actualiza el contador
               if varb["value"] == False:
                    contador = contador + d - 1
          elif a == "goto":
               # Actualiza el contador
               contador = contador + d - 1
          elif a == "return":
               vard = buscaVariable(param,var,d)
               if vard.get("value") == None:
                    print("ERROR: Variable no inicializada")
                    raise CalcError("Expresion invalida")
               # Regresa el valor de la variable
               return vard["value"]
          elif a == "write":
               vard = buscaVariable(param,var,d)
               if vard.get("value") == None:
                    print("ERROR: Variable no inicializada")
                    raise CalcError("Expresion invalida")
               # Imprime el valor de la variable
               print(vard["value"])
          elif a == "callf":
               tempValues = []
               # Crea un listado con los valores de los parametros recibidos
               for value in c:
                    varc = buscaVariable(param,var,value)
                    tempValues.append(varc["value"])
               # Llama a ejecutar la funcion especial
               callf(b,tempValues)
          else:
```

```
                print("ERROR: CALL01")
                raise CalcError("Error en sistema")
            contador = contador + 1
    return "Sys None"
```

Parámetros

function: nombre de la función a ejecutar.

param: parámetros de la función que se ejecuta.

var: variables de la función que se ejecuta.

Retorno

Valor de retorno de la función o el string “Sys None” en caso de no llegar a un return.

Op

Función

Regresa el resultado de la operación enviada.

Código

```
def op(op,opdo1,opdo2):
    if op == '*':
        return opdo1 * opdo2
    elif op == '/':
        return opdo1 / opdo2
    elif op == '-':
        return opdo1 - opdo2
    elif op == '+':
        return opdo1 + opdo2
    elif op == '==':
        return opdo1 == opdo2
    elif op == '<':
        return opdo1 < opdo2
    elif op == '>':
        return opdo1 > opdo2
    elif op == '<=':
        return opdo1 <= opdo2
    elif op == '>=':
        return opdo1 >= opdo2
    elif op == '!=':
        return opdo1 != opdo2
```

```
    elif op == '&':
        return opdo1 and opdo2
    elif op == '|':
        return opdo1 or opdo2
    else:
        print("ERROR: OP01")
        raise CalcError("Error en sistema")
```

Parámetros

op: signo de la ecuación a ejecutar.

opdo1: primer número de la ecuación.

opdo2: segundo numero de la ecuación.

Retorno

Resultado de la ecuación.

BuscaVariable

Función

Busca la variable especificada en *var* entre las variables globales y locales.

Código

```
def buscaVariable(temparam,tempvar,var):
    # Busca la variable en entre las variables globales
    if variables["main"]["var"].get(var) != None:
        return variables["main"]["var"][var]
    elif variables["active sys"] != "main":
        # Si la funcion activa no es main, busca entre los parametros y va
riables locales recibidos
        if tempvar.get(var) != None:
            return tempvar[var]
        elif temparam.get(var) != None:
            return temparam[var]
    print("ERROR: Variable no encontrada")
    raise CalcError("Variable invalida")
```

Parámetros

temparam: apuntador a los parámetros de la función activa.

tempvar: apuntador a las variables de la función activa.

var: nombre de la variable a buscar.

Retorno
Apuntador a la variable encontrada.

BuscaTipo

Función
Busca el tipo del dato recibido, ya sea una constante o una variable.

Código
```
def buscaTipo(tipo):
     if type(tipo) == type(1):
          return "int"
     elif type(tipo) == type(1.0):
          return "float"
     elif tipo[0] == "'":
          return "char"
     elif tipo[0] == '"':
          return "str"
     elif (tipo == "true" or tipo == "false"):
          return "bool"
     elif variables["main"]["var"].get(tipo) != None:
          return variables["main"]["var"][tipo]["type"]
     elif variables["active sys"] != "main":
          if variables[variables["active sys"]]["var"].get(tipo) != None:
               return variables[variables["active sys"]]["var"][tipo]["type"
]
          elif variables[variables["active sys"]]["param"].get(tipo) != None
:
               return variables[variables["active sys"]]["param"][tipo]["typ
e"]
     print("ERROR: Tipo de variable no encontrada")
     raise CalcError("Variable invalida")
```

Parámetros
tipo: constante o variable a la cual buscar el tipo de dato.

Retorno
Tipo de dato encontrado.