



Tecnológico de Monterrey

Documentación Final de Proyecto

Francisco Salgado Guízar A01365047

05 de junio del 2023

Diseño de Compiladores

Elda Guadalupe Quiroga González

Lenguaje de programación W	2
Descripción y Documentación Técnica del Proyecto	2
Descripción del proyecto	2
Propósito y Alcance del proyecto	2
Análisis de requerimientos y principales casos de prueba	2
Proceso general seguido para desarrollo de proyecto	5
Descripción del lenguaje	10
Listado de errores que pueden ocurrir	10
Descripción del compilador	13
Equipo de cómputo, lenguaje y utilerías especiales usadas en el proyecto	13
Descripción del análisis léxico	14
Descripción de análisis sintáctico	16
Descripción de Generación de Código Intermedio y Análisis Semántico	17
Códigos de operación para el código intermedio	17
Diagramas de sintaxis y puntos neurálgicos	19
Tabla de consideraciones semánticas	28
Descripción detallada del proceso de Administración de Memoria usado en compilación	30
Descripción de la máquina virtual	33
Pruebas del funcionamiento del lenguaje	33
Factorial cíclico	33
Factorial recursivo	35
Bubble Sort	36
Find en arreglo	39
Multiplicación de matrices 3x3	44
Documentación del código del proyecto	49
Función addArray en línea 931 del archivo compiler.py	49
Función addMatrix en línea 1045 del archivo compiler.py	49
Función pushTemporal en línea 1422 del archivo compiler.py	49
Función addVariable en línea 1453 del archivo compiler.py	50
Función addFunc en línea 13 del archivo FuncTable.py	50
Función getVarAddress en línea 23 del archivo FuncTable.py	50
Función addQuad en línea 8 del archivo QuadList.py	50
Función addVar en línea 9 del archivo VarsTable.py	50
Función searchVar en línea 44 del archivo VarsTable.py	50
Función setValue en línea 12 del archivo VirtualMemory.py	50
Función getValue en línea 23 del archivo VirtualMemory.py	51
Manual del usuario	51

Lenguaje de programación W

Descripción y Documentación Técnica del Proyecto

Descripción del proyecto

Propósito y Alcance del proyecto

El propósito de este proyecto es proporcionar a programadores de diferentes niveles de expertise las herramientas necesarias para llevar a cabo la creación de código para ejecutar operaciones aritméticas, código reusable a través de módulos y una forma estructurada de elementos atómicos en forma de arreglos y matrices. De la misma manera, el lenguaje podrá soportar la implementación de data frames y funciones base para el análisis estadístico de varios elementos precargados en un archivo exterior.

Análisis de requerimientos y principales casos de prueba

Los principales requerimientos del programa se pueden dividir en sus funciones e implementaciones más esenciales que son:

1. Reglas intuitivas para la escritura de código fácilmente entendible.
2. Análisis léxico y sintáctico que marque errores si la estructura del programa y los estatutos en el código son incorrectos.
3. Análisis semántico sobre los resultados de la ejecución de operaciones aritméticas y sus respectivos tipos, así como mantener la consistencia de los tipos atómicos de datos a través de remarcar errores al momento de compilación.
4. Ejecución de diferentes estatutos lineales, asignaciones y operaciones aritméticas que se resuelven de izquierda a derecha (Leftmost derivation).
5. Ejecución de estatutos no lineales y condicionados como el if, else, for y while.
6. Declaración de variables de diferente tipo atómico y su respectivo análisis semántico.
7. Declaración de módulos o funciones que pueden regresar un resultado al terminar su ejecución, así como sus respectivas declaraciones de variables locales y parámetros.
8. Llamadas a funciones como expresiones, si es que la función regresa un resultado, o sólo para ejecución de código recurrente o reciclado.
9. Declaración de arreglos y matrices de tamaño fijo que permita acceder y asignar a sus elementos a través de expresiones que representen sus índices.
10. Creación de un código objeto que contendrá toda la información relevante para la ejecución como las direcciones de memoria de constantes y sus valores, el tamaño de memoria que utilizarán las funciones y una lista de cuádruplos en orden que representarán el orden de las instrucciones.
11. La ejecución del código objeto a través de una máquina virtual que maneja la memoria y ejecutará en orden las instrucciones representadas en cuádruplos.
12. La asignación de variables a través de consola y tener la capacidad de imprimir los resultados como expresiones.

Para simplificar el análisis de requerimientos, se llevaron a cabo una serie de casos de prueba para comprobar el funcionamiento correcto del programa, entre los cuáles los más importantes fueron:

Test Cases			
Descripción	Pasos	Información de prueba	Resultados
Archivo del código tiene terminación .w	1- Ejecutar el script “compilador.py”. 2- Poner el nombre del archivo que contiene el código. 3- Obtener resultado de la compilación.	prueba.w	Compilación ejecutada con éxito y creación del código intermedio identificado con el nombre “obj.a”.
Estructura base del código para compilar	1- Poner la declaración de variables globales (opcional). 2- Poner la declaración de módulos (opcional). 3- Poner la función principal desde donde iniciará la ejecución del programa identificada por “main”. 4- Terminar el programa con la palabra “end”.	<pre>Unset main { } end</pre>	Compilación exitosa
Declarar una variable de tipo atómico	1- Poner la palabra “variable” 2- Poner el tipo atómico 3- Poner el nombre de la variable 4- Poner el nombre de una variable del mismo tipo seguida de una coma 5- Terminar con “;”	<pre>Unset var int x, y, z; var float a,b;</pre>	Compilación exitosa
Declarar una función con sus parámetros	1- Poner la palabra “func” 2- Poner el tipo de retorno de la función 3- Declarar parámetros		Compilación exitosa

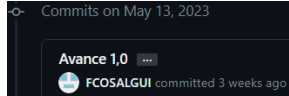

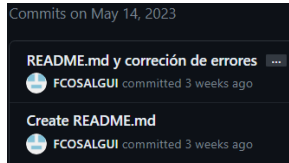
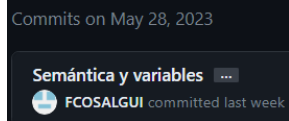
	entre“()” 4- Declarar variables locales	<pre>Unset func int modulo(int x, float a) var string palabra; { return x+1; }</pre>	
Hacer ejecución de estatutos lineales y asignar resultado	1- Declarar variables 2- Hacer una expresión aritmética 3- Asignar resultado de expresión a una variable	<pre>Unset var int resultado; var int x, y, z; main { resultado = (x * y) + z; } end</pre>	Compilación exitosa
Crear un estatuto no lineal tipo if	1- Poner “if” 2- Poner la condición entre “()” 3- Crear el cuerpo entre “{ }” 4- Agregar else con su respectivo cuerpo	<pre>Unset if(a > b){ c = a; } else { c = b; }</pre>	Compilación exitosa
Crear un ciclo while	1- Poner “while” 2- Poner la condición entre “()” 3- Crear el cuerpo entre “{ }”	<pre>Unset while(a != true){ resultado = b + c; }</pre>	Compilación exitosa
Crear ciclo for	1- Poner “for” 2- Poner la asignación de		Compilación exitosa

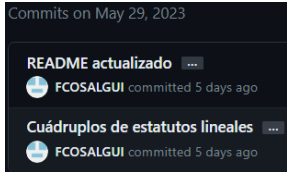
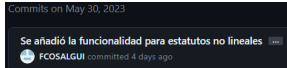
	la variable de control 3- Poner el aumento de la variable de control como asignación 4- Poner condición para ejecutar el ciclo 5- Poner el cuerpo del ciclo entre “{ }”	<pre>Unset for(i = 0; i = i+1; i < n) { }</pre>	
Creación de arreglos y matrices	1- Poner “var” 2- Poner el tipo de arreglo 3- Asignar tamaños con constantes de tipo entero	<pre>Unset var int array[10]; var bool matrix [3][5];</pre>	Compilación exitosa
Acceso a elementos de variables y matrices	1- Poner el nombre del arreglo 2- Poner el índice en formato de expresión	<pre>Unset array[0] = matrix[j+i][array[i+ 4]];</pre>	Compilación exitosa
Llamadas a funciones	1- Poner el nombre de la función 2- Poner parámetros entre “()”	<pre>Unset resultado = modulo(4, n-1);</pre>	Compilación exitosa

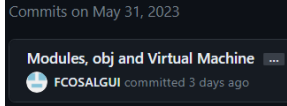

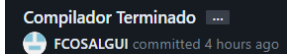
Proceso general seguido para desarrollo de proyecto

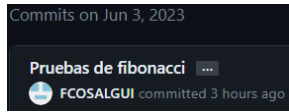
El proyecto se fue desarrollando por medio de avances y funcionalidades como en el orden que eran vistas en clase. Por ejemplo, como primero se vió el tema de cómo ejecutar estatutos lineales por medio del uso de varias pilas y la generación de cuádruplos antes de la generación de código intermedio para estatutos no lineales, primero se desarrolló la creación de código intermedio para estatutos lineales antes que la de estatutos no lineales.

El orden de las funcionalidades desarrolladas, así como los días en que fueron terminadas y la lista de commitments es la siguiente:

Bitácora			
Avance	Bitácora	Commit	Día del commit
Análisis léxico y sintáctico	Tomó bastante tiempo terminar esta parte del proyecto por investigar bien cómo utilizas PLY y asignar los Tokens correctamente, aparte de querer tener una base sólida para el resto del proyecto.		13 de mayo de 2023
Corregir errores de gramática	Hubo varios problemas al momento de diseñar la gramática. Por ejemplo, se creaba una recursión infinita en una de las generaciones de un no terminal, y estuve explorando cómo implementar los puntos neurálgicos por medio de PLY.		14 de mayo de 2023
README.md creado	He empezado a desarrollar el manual de usuario a través de un README dentro del repositorio, ya que me permite dividir de forma muy fácil las secciones para poner ejemplo del código.		14 de mayo de 2023
Análisis semántico e implementación de direcciones virtuales	Ha pasado un buen rato desde que trabajé en el proyecto y, tomando en cuenta lo que aprendí en clase, aplicaré lo que aprendí sobre usar		28 de mayo de 2023

	<p>direcciones virtuales para implementar la creación del código intermedio por medio de cuádruplos. Aparte he logrado que se puedan guardar direcciones en la tabla de variables de la función global o “main”. Después implementaré que se puedan guardar de forma local.</p>		
Cuádruplos de estatutos lineales	<p>Esta semana será la más pesada de todas, definitivamente me costará mucho acabar las características principales del proyecto, pero por lo menos he logrado que el código intermedio se componga de cuádruplos de estatutos lineales y asignaciones, todo basado en direcciones virtuales, incluyendo los operadores.</p>		29 de mayo de 2023
Estatutos no lineales	<p>Logré implementar la creación de cuádruplos basados en estatutos no lineales utilizando una pila de saltos. Ahora se pueden declarar ciclos y estatutos condicionados, lo cual le agrega cada vez más funcionalidad al programa.</p>		30 de mayo de 2023

Creación de módulos, llamadas y primera versión de la máquina virtual.	Siendo este el avance más crucial de todo el proyecto, he logrado implementar el poder declarar funciones con parámetros, variables y temporales locales. Además, se pueden llamar y si tienen un retorno, sus resultados se pueden usar como expresiones. Después de esto me dediqué a hacer la primera versión de la máquina virtual para probar la generación de código intermedio antes y arreglar cualquier error que no haya visto.	 <p>Commits on May 31, 2023</p> <p>Modules, obj and Virtual Machine</p> <p>FCOSALGUI committed 3 days ago</p>	31 de mayo de 2023
Arreglos y ejecución del código intermedio	Me costó varios días y mucho resolver bugs y errores, pero finalmente logré crear una versión funcional del compilador y la máquina virtual con su memoria virtual. Empecé a realizar las pruebas obligatorias para verificar el funcionamiento del programa y, aunque han sido exitosas, creo que requieren de mayores pruebas para que todo funcione bien.	 <p>Primera versión terminada y primeras pruebas</p> <p>FCOSALGUI committed 14 hours ago</p>	03 de junio
Compilador funcionando con recursión,	He logrado que las pruebas de factorial, bubble sort, y find	 <p>Compilador Terminado</p> <p>FCOSALGUI committed 4 hours ago</p>	03 de junio de 2023

condicionales y funciones	<p>sean todo un éxito. Tuve que corregir varias cosas todavía sobre escenarios que no había previsto pero finalmente puedo decir que puedo escribir un programa en el lenguaje. El mayor problema que tengo ahora tiene que ver con la declaración de strings directamente en el código, lo que causa que me sea difícil guardarlos al momento de ejecución, pero lo más importante ha quedado.</p>		
Últimas pruebas y correcciones	<p>Después de realizar las últimas pruebas y verificar que mi programa es capaz de ejecutar cosas tan complejas como una multiplicación de matrices con éxito, me siento muy feliz con los resultados que he obtenido. Desgraciadamente, hay muchas funcionalidades que no podré ser capaz de implementar por falta de tiempo y porque quiero dedicarme a hacer una documentación de calidad.</p>		03 de junio de 2023

Después de terminar este proyecto puedo decir que he aprendido lecciones muy valiosas tanto para mi vida profesional como para mi vida personal. Entre ellas se encuentran el hecho de que tengo que ser más organizado con mis tiempos y no subestimar la carga de trabajo de un proyecto. Ciertamente pude haber hecho un mejor trabajo de haber tenido una mayor disciplina para desarrollar este proyecto final, pero estoy satisfecho con el resultado. Aprendí muy a la mala que no dedicarle su tiempo a las cosas tiene sus consecuencias, y el poco tiempo que he dormido estas últimas semanas debido a toda la carga de trabajo que me impuse para tratar de entregar un proyecto de calidad me ha enseñado a cuidar más de mí y a ser más responsable.

Descripción del lenguaje

El lenguaje W es un lenguaje de programación orientado al análisis estadístico que ofrece las herramientas que un programador necesita para crear código de forma intuitiva. Inspirado en lenguajes de programación como C para la declaración de variables, funciones y el orden de ejecución del código, W es un lenguaje cuyo nivel de abstracción únicamente permite la implementación de funciones o módulos más no es un lenguaje de programación orientado a objetos.

El lenguaje puede ejecutar estatutos lineales como asignaciones y resolver expresiones aritméticas. También se pueden declarar variables de tipo enteras, flotantes, strings de caracteres, y variables booleanas. Se pueden ejecutar estatutos condicionales con if y else y también se pueden crear ciclos con for y while. Se pueden crear funciones del mismo tipo atómico que las variables y de tipo void que no regresará ningún resultado. También se pueden declarar arreglos y matrices de tamaño fijo para asignarles valor a cada elemento indexado y acceder a sus elementos a través de expresiones.

Listado de errores que pueden ocurrir

Al momento de llevar a cabo el proceso de compilación o de ejecución se pueden encontrar errores que detienen cualquiera de ambos procesos para indicarle al usuario qué error se encontró.

Errores en compilación		
Tipo de error	Razones del error	Mensaje de error
Syntax Error	No se siguió la sintaxis correcta del programa	<pre>Syntax error at {token} on line {line number} of type {type}</pre>
Void type return error	Una función de tipo “void” tiene un un “return” al final de su expresión cuando una función void no regresa nada	<pre>Error: The void function + functionName + has a return expression</pre>

Function type return error	El valor después de “return” al final de una función no es del mismo tipo que el tipo de la función	Error: Return expression for function + functionName + is not the same as the function type
Return not declared	Al final de una función que no es de tipo void no se ha declarado el valor para regresar al final de su ejecución	Error: Return expression for function + functionName + is not declared
Calling a function that does not exist	Se está tratando de mandar a llamar una función que no ha sido declarada	Error: Function ID with name + functionName + not declared
Invalid parameter type	Se está tratando de enviar un valor a un parámetro cuyo tipo no coincide con el del parámetro	Error: Type of argument sent to parameter of function + funcName + doesn't coincide
Exceeding number of parameters	El total de parámetros enviados a una llamada de función excede la cantidad de parámetros que la función acepta	Error: Tried to send more parameters to function + funcName + than it accepts
Not enough parameters sent	El total de parámetros enviados a una llamada de función es menor a la cantidad de parámetros que acepta la función	Error: Tried to retrieve expression to add to parameter but the operand stack is empty
Incorrect number of parameters	El número de parámetros enviados a una llamada de función no coincide con los que la función acepta	Error: Parameters sent to function + funcName + does not coincide with the number that it accepts

Stack overflow	La cantidad de variables, constantes y temporales son más que los que puede guardar la memoria en su programa	<code>Stack overflow of + type for " + identifier</code>
Invalid expression for indexing	La expresión usada para indexar un arreglo no es de tipo entera	<code>Expression sent to index array + arrayName + is not integer. You can only index arrays with integers</code>
Invalid type for assignment	La expresión que se asignará a una variable no es del mismo tipo que la variable	<code>Type mismatch when assigning a value to a variable for address: + address</code>
Invalid expression for conditional if	La expresión usada para entrar a un condicional "if" no es de tipo booleana	<code>Type mismatch at if evaluation, it is not a boolean</code>
Invalid expression for conditional at while loop	La expresión usada para condicionar un "while" no es válida	<code>Type mismatch at while evaluation, it is not a boolean"</code>
Invalid expression for conditional at for loop	La expresión usada para condicionar un "for" no es válida	<code>Type mismatch at for boolean component, it is not boolean</code>
Invalid file name	El nombre del archivo a compilar no tiene terminación .w	<code>Error: Nombre del archivo no tiene terminacion .w y no se compilara</code>
Function ID already declared	Se trató de crear una función con el mismo identificador que otra existente	<code>Function + name + has already been declared</code>
Variable ID already declared	Se trató de declarar una	<code>Variable " + name + "</code>

	función con el mismo identificador que otra existente	<code>has already been declared</code>
ID not found	Se ha tratado de usar una variable que no ha sido declarada	<code>Variable + name + has not been declared</code>

Errores en ejecución		
Tipo de error	Razones del error	Mensaje de error
Stack overflow of modules	Se han hecho muchas llamadas a funciones dentro de función y estas han excedido la capacidad de memoria del programa	<code>Error: Stack Overflow of memory for modules</code>
Array out of bounds	Se ha intentado acceder un índice de un arreglo que está fuera de los límites definidos para dicha variable	<code>Array out of bounds</code>
Value not assigned	Se ha tratado de usar una variable sin un valor asignado para un estatuto	<code>Address {address} does not have a value assigned to it\nMake sure all variables have values assigned to them in your code!</code>

Descripción del compilador

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

Se utilizó el lenguaje de programación Python en su versión 3.11.3 para el 100% de la realización de este proyecto, utilizando de herramienta para análisis léxico y sintáctico PLY en su versión 3.11. Además, todo el proyecto fue hecho en el IDE Visual Studio Code y el repositorio fue hospedado en GitHub.

El equipo de cómputo utilizado fue una laptop con un procesador Intel Core i7 de séptima generación, 16 GB de RAM y el sistema operativo Windows 10.

Descripción del análisis léxico

Elementos principales		
Tipo del token	Nombre del token	Expresión regular
Identificador	ID	[a-zA-Z_][a-zA-Z0-9_]*
Constante entera	CTEINT	[-]?[0-9]+
Constante flotante	CTEFLOAT	[+-]?[0-9]+\.[0-9]+(e[-+]?[0-9]+\.[0-9]+)?
Constante de tipo carácter	CTECHAR	\('[^']*' "[^"]*"')\'
Constante de strings	CTESTRING	"([^\n] (\. \n))*?"
Salto de línea	newLine	\n+
And lógico	AND	\&\&
Or lógico	OR	\
Less than para comparación	LT	\<
Greater than para comparación	GT	\>
Less than or equal para comparación	LTOE	\< =\
Greater than or equal para comparación	GTOE	\> =\
Equal para comparación	E	\# =\
Not equal para comparación	NE	\! =\
Corchete izquierdo	LEFTBRACKET	\[
Corchete derecho	RIGHTBRACKET	\]
Paréntesis izquierdo	LEFTPARENTHESIS	\(
Paréntesis derecho	RIGHTPARENTHESIS	\)
Llave izquierda	LEFTCURLYBRACE	\{
Llave derecha	RIGHTCURLYBRACE	\}

Punto y coma	SEMICOLON	\;
Coma	COMMA	\,
Signo de suma	PLUS	\+
Signo de resta	MINUS	\-
Signo de multiplicación	MULTIPLICATION	*
Signo de división	DIVISION	\
Signo igual	EQUAL	\=
Punto	DOT	\.

Palabras reservadas	
Palabra reservada	Token
var	VAR
func	FUNC
void	VOID
int	INT
float	FLOAT
char	CHAR
string	STRING
bool	BOOL
dataframe	DATAFRAME
if	IF
else	ELSE
while	WHILE
for	FOR
read	READ

write	WRITE
return	RETURN
true	TRUE
false	FALSE
end	END
push	PUSH
keys	KEYS
avg	AVG
count	COUNT
max	MAX
min	MIN
sum	SUM
plot	PLOT
histogram	HISTOGRAM
print	PRINT
main	MAIN

Descripción de análisis sintáctico

Gramática formal para representar estructuras sintácticas		
Tipo de estructura	Representación	Ejemplos
Símbolos no terminales	Todo identificador escrito con minúsculas y guiones bajos	exp call_exp var_dec2 modules
Símbolos terminales	Todo identificador escrito en mayúsculas	MAIN FOR ID LEFTBRACKET

Puntos neurálgicos	Todo identificador que empiece con “np_” y esté escrito en minúsculas	np_set_scope_main np_push_for np_end_module
--------------------	---	---

Descripción de Generación de Código Intermedio y Análisis Semántico

Códigos de operación para el código intermedio

Código de operación puesto al comienzo de un cuádruplo		
Símbolo de la operación	Código de identificación	Descripción
<	1	Compara 2 elementos y guarda el resultado de la comparación en un temporal
>	2	Compara 2 elementos y guarda el resultado de la comparación en un temporal
<=	3	Compara 2 elementos y guarda el resultado de la comparación en un temporal
>=	4	Compara 2 elementos y guarda el resultado de la comparación en un temporal
#=	5	Compara 2 elementos y guarda el resultado de la comparación en un temporal
!=	6	Compara 2 elementos y guarda el resultado de la comparación en un temporal
+	7	Suma 2 elementos y guarda el resultado de la comparación en un temporal
-	8	Resta 2 elementos y guarda el resultado de la comparación en un temporal
/	9	Divide 2 elementos y guarda el resultado de la

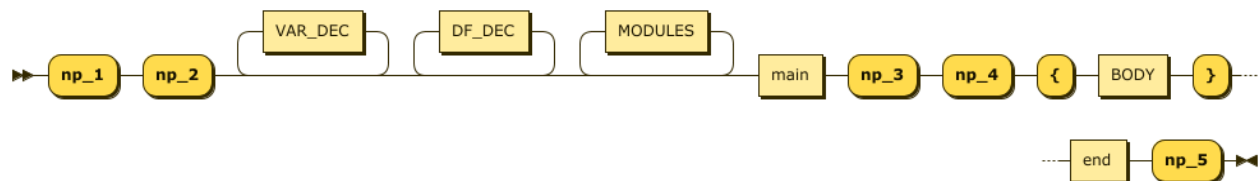
		comparación en un temporal
*	10	Multiplica 2 elementos y guarda el resultado de la comparación en un temporal
=	11	Guarda el valor de una expresión en la dirección especificada
goto	12	Va a la dirección del cuádruplo especificada
gotoF	13	Toma el valor de la expresión booleana y de ser falsa, va a la dirección especificada
gotoV	14	Toma el valor de la expresión booleana y de ser verdadera, va a la dirección especificada
read	15	Guarda el valor que lee en consola a la variable especificada
write	16	Imprime en consola el valor de la expresión que recibe
return	17	Asigna el valor de retorno a una función que no sea de tipo void
endfunc	18	Indica el final de la ejecución de una función
era	19	Asigna el espacio en la memoria virtual para la ejecución de una función
parameter	20	Identifica el resultado de una expresión para ser enviado como parámetro de una función
gosub	21	Va a la dirección de la función a la que se le asignó espacio en la memoria virtual

ver	22	Verifica que la expresión usada para indexar un arreglo esté entre el rango de sus dimensiones
end	23	Indica el fin de ejecución del programa
+dirb	24	Suma el resultado de una expresión de asignación con la dirección base del arreglo para obtener la dirección en memoria

Diagramas de sintaxis y puntos neurálgicos

“np” Es el símbolo dentro de los diagramas que representa los puntos neurálgicos y cada uno viene con un identificador único en forma de un número. Cada punto neurálgico de cada diagrama tiene una explicación de las acciones que realiza.

PROGRAM::=



np_1: Crea la función de “main” y la anexa en la tabla de funciones. Se crea el cuádruplo que indicará donde empieza el flujo principal del programa.

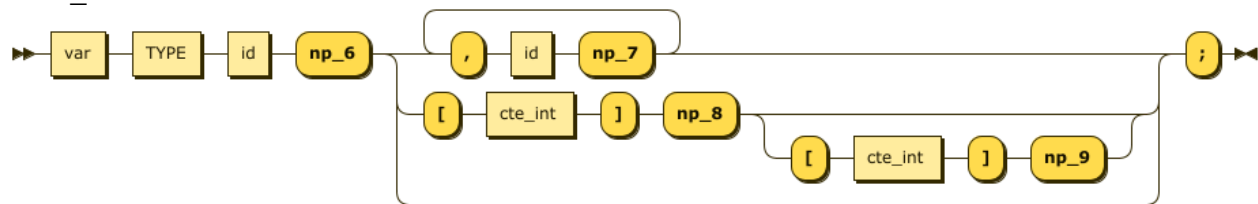
np_2: Cambia el scope de las variables a declararse a “main”.

np_3: Cambia el scope a “main” nuevamente y se le asigna al cuádruplo de “goto” de “main” la dirección del cuádruplo del inicio del flujo del programa principal.

np_4: Se ponen los contadores de los temporales en 0 para asignar temporales globales a los resultados de los estatutos.

np_5: Crea el último cuádruplo que indica el final del programa.

VAR_DEC::=



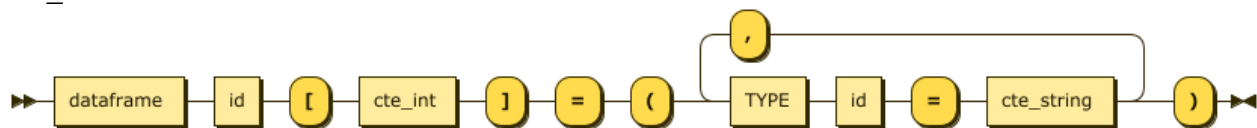
np_6: Agrega la variable a la tabla de variables de su respectivo “scope”, ya sea global o local con el tipo definido.

np_7: Agrega la variable a la tabla de variables de su respectivo “scope” con el mismo tipo de variable definido anteriormente.

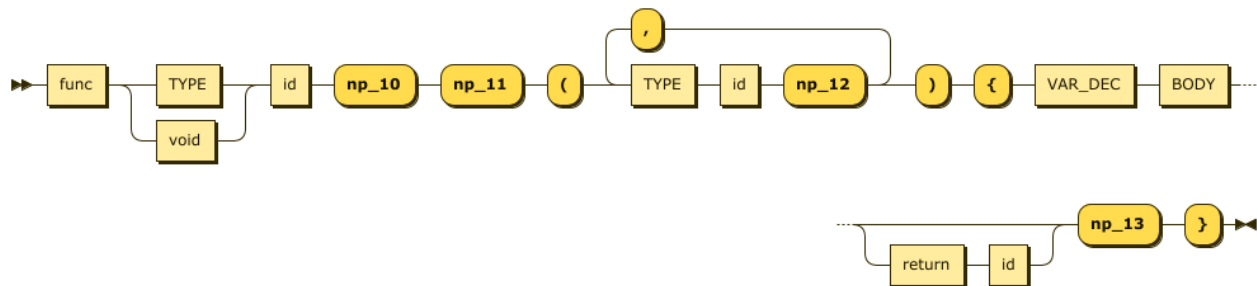
np_8: Agrega el arreglo a la tabla de variables con su dimensión y el cálculo respectivo del espacio que utilizará de direcciones dependiendo de su tipo.

np_9: Agrega la matriz a la tabla de variables con su dimensión y el cálculo respectivo del espacio que utilizará de direcciones dependiendo de su tipo.

DF_DEC ::=



MODULE ::=



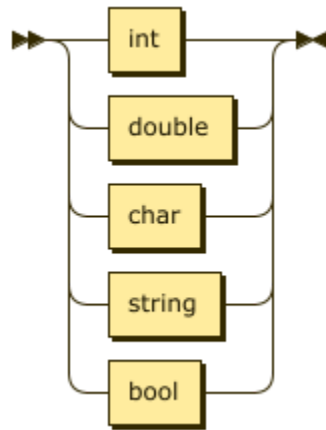
np_10: Crea la función y la anexa en la tabla de funciones ya con su tipo definido y su tabla de variables.

np_11: Cambia el “scope” en el cuál se generarán los temporales y se guardarán las variables declaradas de forma local.

np_12: Agrega el parámetro a la tabla de variables de la función y la guarda en una lista de parámetros que se utilizará para saber cuántas variables se deben enviar a la función.

np_13: Al final, se verifica que exista un return como último estatuto de la función en caso de que sea de tipo diferente de void y vuelve a poner los contadores de los temporales y locales en 0.

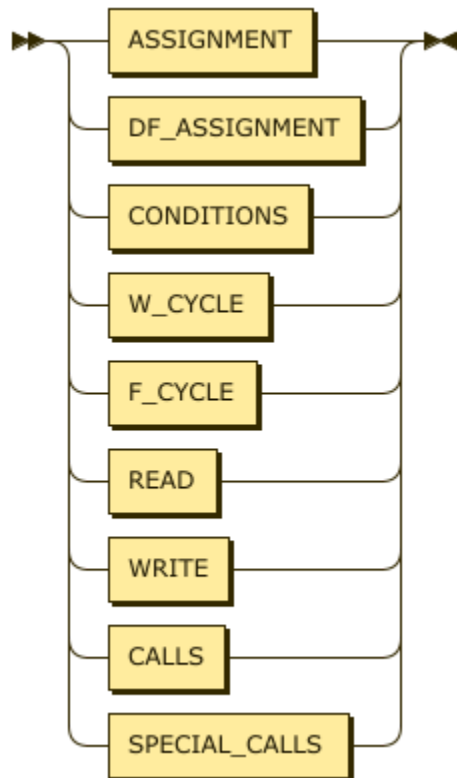
TYPE::=



BODY::=



STATEMENT::=



ASSIGNMENT::=

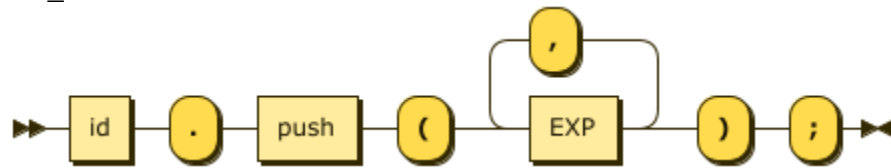


np_14: Determina si la variable es atómica o estructurada y hace los cálculos y validaciones para obtener su dirección y empujarla a la pila de operandos.

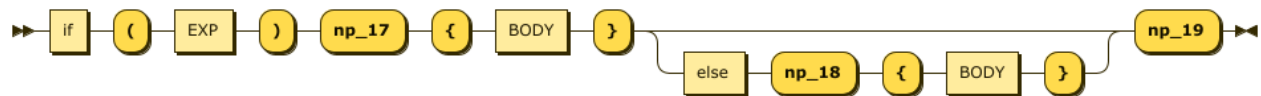
np_15: Empuja el signo de igual a la pila de operadores.

np_16: Saca el tope de la pila de operadores y de ser este el signo de igual, saca los primeros 2 operandos de la pila con su respectivo tipo y crea el cuádruplo de asignación.

DF_ASSIGNMENT::=



CONDITION::=



np_17: Saca el primer elemento de la pila de operandos y su tipo. Determina si el tipo del operando es booleano y de ser así, crea el cuádruplo de “gotof” y mete la dirección del cuádruplo a la pila de saltos.

np_18: Crea un cuádruplo de tipo “goto”. Luego saca el tope de la pila de saltos para asignarle el valor del siguiente cuádruplo a esa dirección y empuja el cuádruplo del “goto” a la pila de saltos.

np_19: Saca el tope de la pila de cuádruplos y le asigna la dirección del final del condicional.

W_CYCLE::=

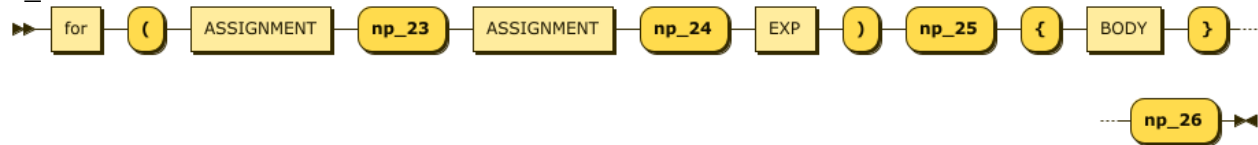


np_20: Empuja la dirección del último cuádruplo creado a la pila de saltos.

np_21: Saca el primer elemento de la pila de operandos y su tipo. Determina si el tipo del operando es booleano y de ser así, crea el cuádruplo de “gotof” y mete la dirección del cuádruplo a la pila de saltos.

np_22: Saca los 2 primeros elementos de la pila de saltos. Al primero le asigna la dirección del siguiente cuádruplo y con el segundo crea un cuádruplo “goto” que manda a la dirección del segundo elemento.

F_CYCLE



np_23: Crea un cuádruplo de tipo “goto” y mete 2 direcciones a la pila de saltos. La primera es el siguiente cuádruplo y la segunda es el cuádruplo actual.

np_24: Saca el tope de la pila de saltos y a ese “goto” le asigna la dirección del siguiente cuádruplo.

np_25: Saca el primer elemento de la pila de operandos y su tipo. Determina si el tipo del operando es booleano y de ser así, crea el cuádruplo de “gotof” y mete la dirección del cuádruplo a la pila de saltos.

np_26: Saca los 2 primeros elementos de la pila de saltos. Al primero le asigna la dirección del siguiente cuádruplo y con el segundo crea un cuádruplo “goto” que manda a la dirección del segundo elemento.

READ::=

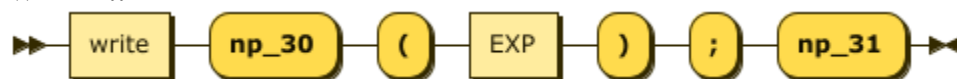


np_27: Empuja el operador de “read” a la pila de operadores.

np_28: Empuja la dirección virtual de la variable a la pila de operandos.

np_29: Saca el tope de la pila de operadores y el tope de la pila de operandos para crear el cuádruplo que ejecuta la instrucción de “read”.

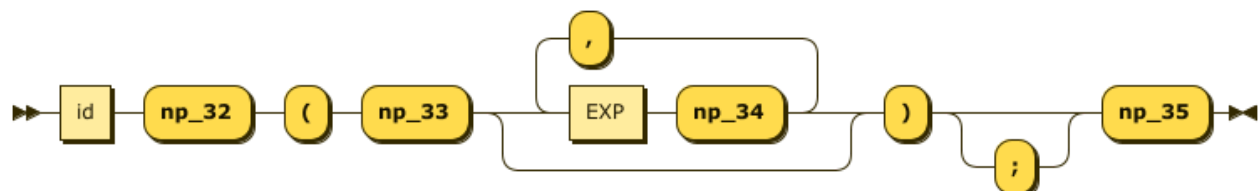
WRITE::=



np_30: Empuja el operador de “write” a la pila de operadores.

np_31: Saca el tope de la pila de operadores y el tope de la pila de operandos para crear el cuádruplo que ejecuta la instrucción de “write”.

CALLS::=



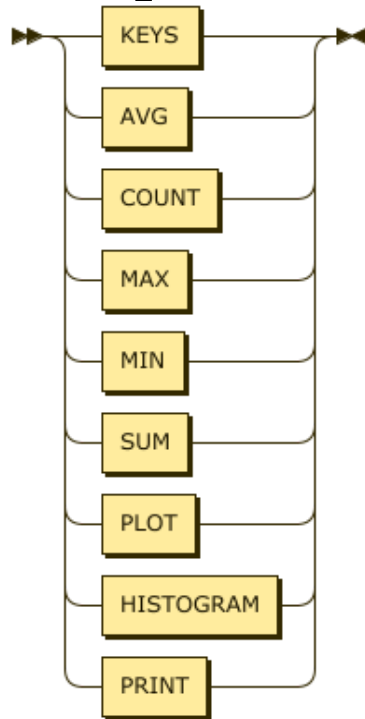
np_32: Verifica que el nombre de la función que se está invocando exista en la tabla de funciones.

np_33: Crea un contador de parámetros y un cuádruplo con la función “era” que indica la dirección del primer cuádruplo de la función que se está invocando.

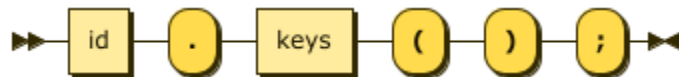
np_34: Se encarga de verificar que la expresión usada para enviar como parámetro cumpla con los requisitos para crear un cuádruplo de tipo “param” que envía guarda el valor de la expresión en la dirección virtual del parámetro.

np_35: Utiliza el contador de parámetros para verificar que la cantidad de parámetros enviados a la función sean correctos. De ser así, crea el cuádruplo de tipo “gosub” con la dirección donde empieza la ejecución de la función.

SPECIAL_CALLS::=



KEYS::=



AVG::=



COUNT::=



MAX::=



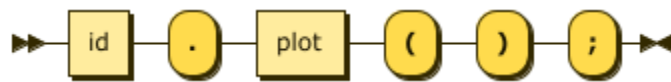
MIN::=



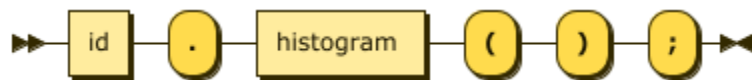
SUM::=



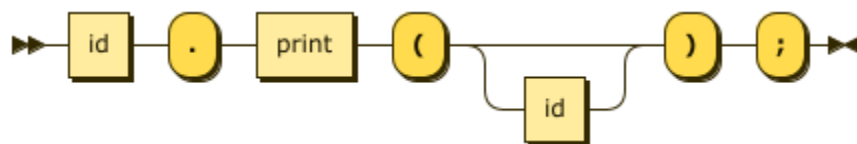
PLOT::=



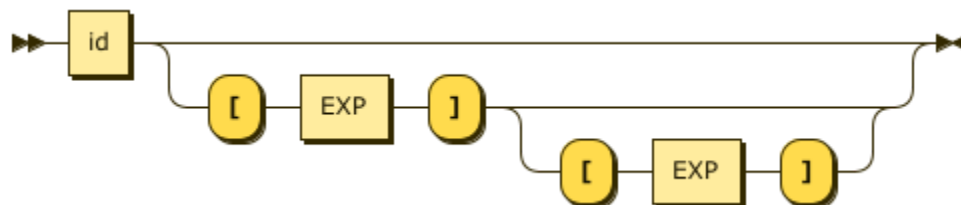
HISTOGRAM::=



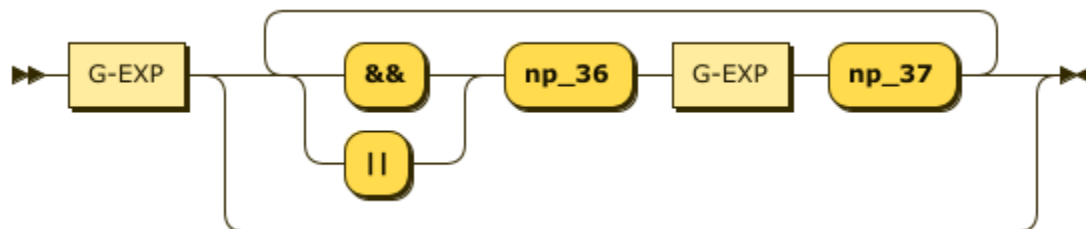
PRINT::=



VARIABLE::=



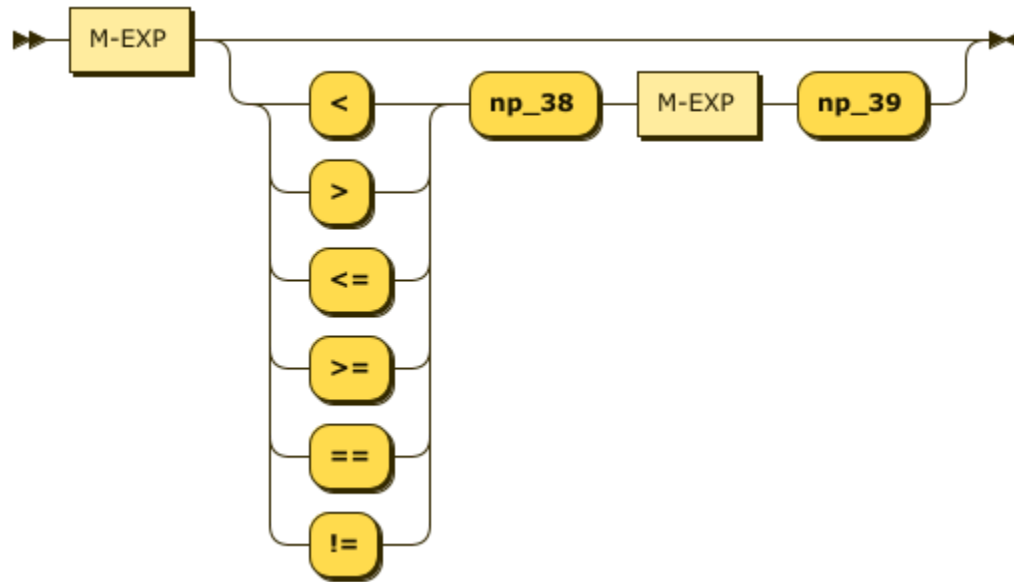
EXP::=



np_36: Empuja el operador a la pila de operadores.

np_37: Si el operador es del tipo “&&” o “||” crea un cuádruplo con esa expresión donde toma los 2 primeros elementos de la pila de operandos, los pasa por la verificación del cubo semántico y luego mete la dirección del resultado temporal a la pila de operandos.

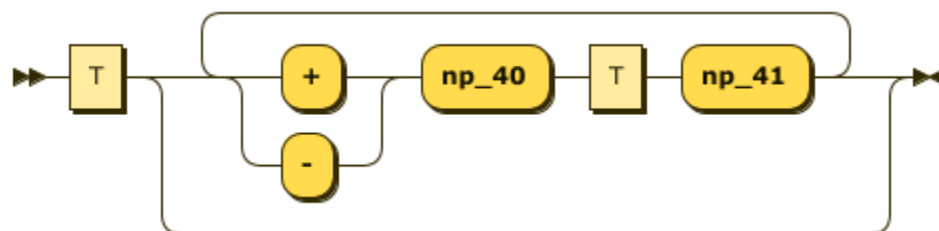
G-EXP::=



np_38: Empuja el operador a la pila de operadores.

np_39: Si el operador es del tipo “<”, “>”, “<=”, “>=”, “<=”, “#”, “!=” crea un cuádruplo con esa expresión donde toma los 2 primeros elementos de la pila de operandos, los pasa por la verificación del cubo semántico y luego mete la dirección del resultado temporal a la pila de operandos.

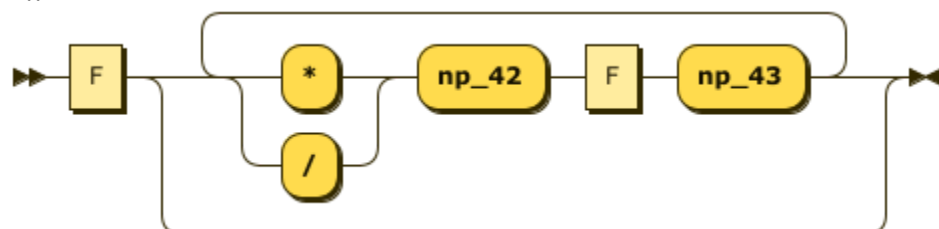
M-EXP::=



np_40: Empuja el operador a la pila de operadores.

np_41: Si el operador es del tipo “+” o “-” crea un cuádruplo con esa expresión donde toma los 2 primeros elementos de la pila de operandos, los pasa por la verificación del cubo semántico y luego mete la dirección del resultado temporal a la pila de operandos.

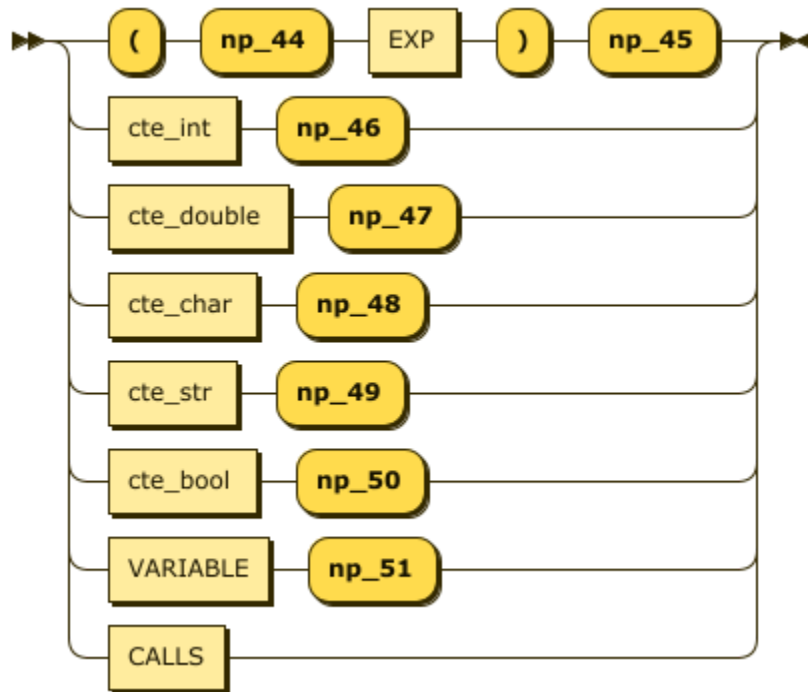
T::=



np_42: Empuja el operador a la pila de operadores.

np_43: Si el operador es del tipo “*” o “/” crea un cuádruplo con esa expresión donde toma los 2 primeros elementos de la pila de operandos, los pasa por la verificación del cubo semántico y luego mete la dirección del resultado temporal a la pila de operandos.

F::=



np_44: Crea un separador temporal identificado como “s” que evita que identifiquen operadores de fuera del paréntesis como parte de la expresión dentro del paréntesis.

np_45: Se elimina el separador temporal después de salir de la expresión del paréntesis.

np_46: Se guarda la constante de tipo entero en la tabla de variables constantes dentro del programa principal con su respectiva dirección virtual y se empuja la constante a la pila de operandos.

np_47: Se guarda la constante de tipo flotante en la tabla de variables constantes dentro del programa principal con su respectiva dirección virtual y se empuja la constante a la pila de operandos.

np_48: Se guarda la constante de tipo carácter en la tabla de variables constantes dentro del programa principal con su respectiva dirección virtual y se empuja la constante a la pila de operandos.

np_49: Se guarda la constante de tipo string en la tabla de variables constantes dentro del programa principal con su respectiva dirección virtual y se empuja la constante a la pila de operandos.

np_50: Se guarda la constante de tipo booleana en la tabla de variables constantes dentro del programa principal con su respectiva dirección virtual y se empuja la constante a la pila de operandos.

np_51: Obtiene la dirección de la variable declarada y la empuja a la pila de operandos.

Tabla de consideraciones semánticas

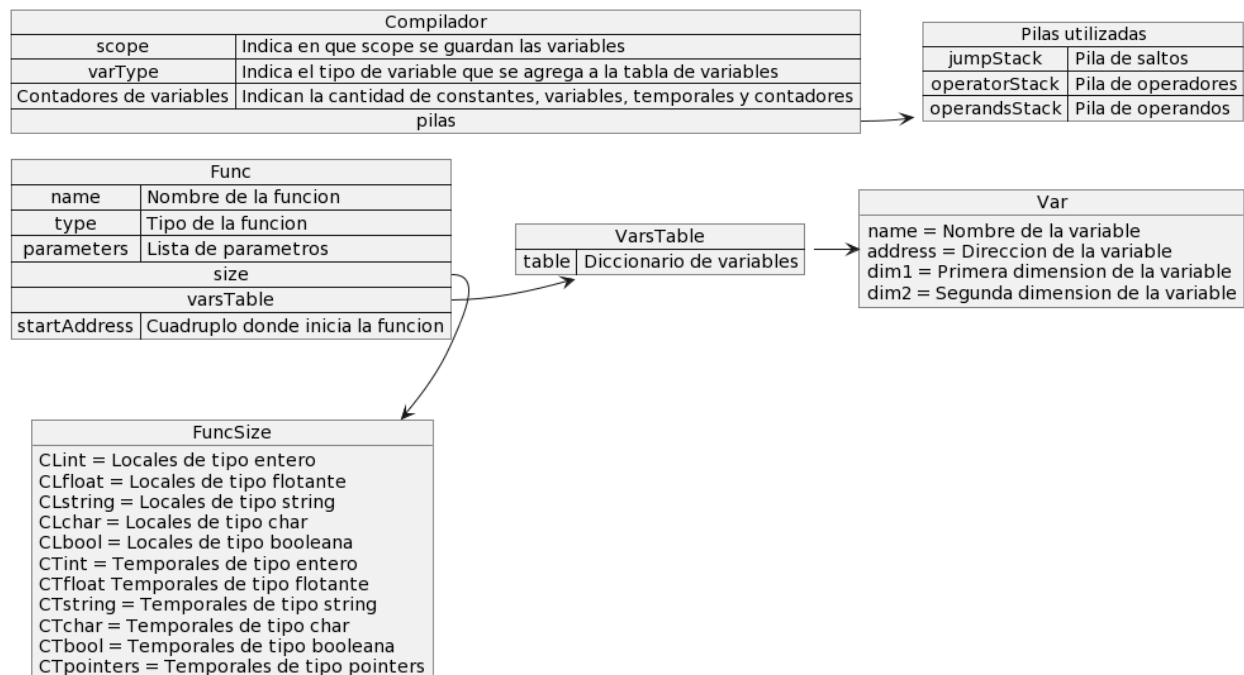
Cubo semántico			
Tipo 1	Tipo 2	Operación	Tipo resultado
int	int	+	int
int	int	-	int
int	int	/	float
int	int	*	int
int	int	&&	bool
int	int		bool
int	int	<	bool
int	int	>	bool
int	int	<=	bool
int	int	>=	bool
int	int	=	int
int	int	#=	bool
int	int	!=	bool
int	float	+	float
int	float	-	float
int	float	/	float
int	float	*	float
int	float	&&	bool
int	float		bool

int	float	<	bool
int	float	>	bool
int	float	<=	bool
int	float	>=	bool
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	=	float
float	float	!=	bool
float	float	!=	bool
float	int	+	float
float	int	-	float
float	int	/	float
float	int	*	float
float	int	&&	bool
float	int		bool
float	int	<	bool
float	int	>	bool

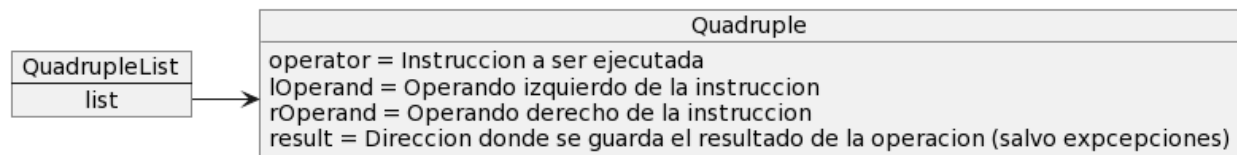
float	int	<=	bool
float	int	>=	bool
string	string	=	string
string	string	#=	bool
string	string	!=	bool
char	char	=	char
char	char	#=	bool
char	char	!=	bool
bool	bool	&&	bool
bool	bool		bool
bool	bool	#=	bool
bool	bool	!=	bool

Descripción detallada del proceso de Administración de Memoria usado en compilación

Compilador o programa main y sus estructuras



Arriba se especifican las estructuras principales del programa y su relación. El compilador hace uso de estas estructuras para generar cuádruplos, cuya estructura es la siguiente:



El compilador manda a llamar a todas estas estructuras para hacer el proceso de compilación y generar el código intermedio. En cuanto a las direcciones virtuales, éstas están definidas por contadores y una lista de variables que indican su inicio y final. Al declararse una variable o generarse un temporal, se utilizan los siguientes contadores y límites para determinar la dirección del elemento:

```

### Direcciones Virtuales ###
# Globales
Gint = [5000, 5999]
Gfloat = [6000, 6999]
Gstring = [7000, 7999]
Gchar = [8000, 8999]
Gbool = [9000, 9999]

# Locales
Lint = [10000, 10999]
Lfloat = [11000, 11999]
Lstring = [12000, 12999]
Lchar = [13000, 13999]
Lbool = [14000, 14999]

# Temporales
Tint = [15000, 15999]
Tfloat = [16000, 16999]
Tstring = [17000, 17999]
Tchar = [18000, 18999]
Tbool = [19000, 19999]

# Constantes
Cint = [20000, 20999]
Cfloat = [21000, 21999]
Cstring = [22000, 22999]
Cchar = [23000, 23999]
Cbool = [24000, 24999]

# Apuntadores de arreglos
Tpointers = [25000, 25999]
  
```



```

### Contadores para variables ###
# Globales
CGint = 0
CGfloat = 0
CGstring = 0
CGchar = 0
CGbool = 0

# Locales
CLint = 0
CLfloat = 0
CLstring = 0
CLchar = 0
CLbool = 0

# Temporales
CTint = 0
CTfloat = 0
CTstring = 0
CTchar = 0
CTbool = 0

# Constantes
CCint = 0
CCfloat = 0
CCstring = 0
CCchar = 0
CCbool = 0

# Apuntadores de arreglos
CTpointers = 0

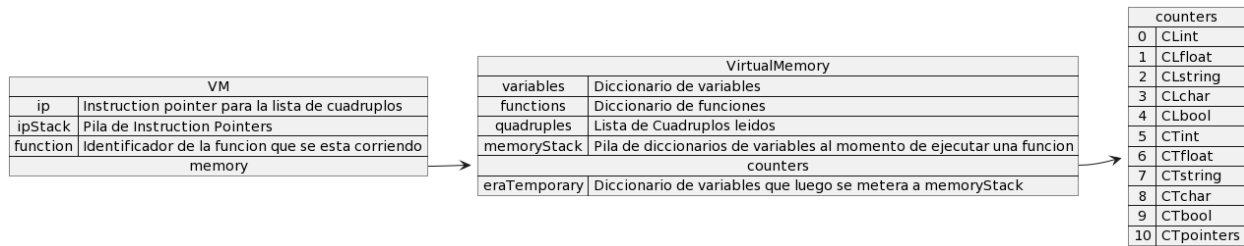
```

Una serie de funciones luego se encarga de asignar las variables, temporales y contadores a sus respectivas funciones o scopes. Al correr instrucciones, se utilizan las funciones de las estructuras de cuádruplos para generarlos, acceder a ellos y modificarlos. Todo esto ocurre conforme se va avanzando a través de la gramática del lenguaje y se accede a sus puntos neurálgicos. Los cuádruplos únicamente guardan direcciones de operadores, de operandos, apuntadores y direcciones de los mismo cuádruplos. No se utilizan literales fuera de lo que guardan las constantes.

Una vez que se ha leído todo el archivo y no se han encontrado errores de ningún tipo, se guardan las constantes, funciones y cuádruplos en un archivo llamado “obj.a”. Este archivo será leído por la máquina virtual y ejecutará el código intermedio en el archivo.

Descripción de la máquina virtual

La máquina virtual accede únicamente a una sola estructura que alberga la funcionalidad para manejar las variables, sus valores, scopes y validaciones. Dicha estructura se ve de la siguiente manera:



La Máquina Virtual o VM se encarga de leer el archivo generado por el compilador y siempre debe tener el nombre de “obj.a”. Al hacerlo, guardará la información contenida en el archivo y empezará la ejecución a través de los operadores de los cuádruplos contenidos en el archivo. De igual manera la Máquina Virtual utiliza una estructura de apoyo que es la Memoria Virtual. Esta se encarga de guardar la información de las variables a través de diccionarios y listas que utiliza la máquina virtual para asignar valores, identificar scopes y obtener valores de las variables al momento de su ejecución, así como direcciones de los cuádruplos y hasta apuntadores para direcciones de arreglos y matrices. Al terminar la ejecución del código intermedio, el programa se detiene.

Pruebas del funcionamiento del lenguaje

Aquí se anexan las pruebas solicitadas para comprobar el funcionamiento e implementación correcta del compilador y la máquina virtual.

Factorial cíclico

Programa en lenguaje W:

```
Unset
var int x, fact, n;
var string lineJump;
main {
    fact = 1;
    write(" Enter a number to find factorial: ");
    read(n);

    for( x = 1 ; x = x + 1 ; x <= n){
        fact = fact * x;
    }
}
```

```

        write(" Factorial of ");
        write(n);
        write(" ");
        write(" is ");
        write(fact);
    }

end

```

Código intermedio (obj.a):

```

Unset
20000 : 1 | 22000 : " Enter a number to find factorial: " | 22001 : " Factorial
of " | 22002 : " " | 22003 : " is " | %%
%%
12,,,1
11,20000,,5001
16,,,22000
15,,,5002
11,20000,,5000
12,,,8
7,5000,20000,15000
11,15000,,5000
3,5000,5002,19000
13,19000,,13
10,5001,5000,15001
11,15001,,5001
12,,,6
16,,,22001
16,,,5002
16,,,22002
16,,,22003
16,,,5001
23,,,
%%

```

Resultado:

```

Enter a number to find factorial: 7
Factorial of 7 is 5040

```

Factorial recursivo

Programa en lenguaje W:

```
Unset
var int x;

func int fact(int y)
var int result;
{
    if (y #= 0){
        result = 1;
    }

    else {
        result = y * (fact(y - 1)) ;
    }

    return result;
}

main {
    write(" Enter a number to find factorial: ");
    read(x);
    write(fact(x));
}

end
```

Código intermedio (obj.a):

```
Unset
20000 : 0 | 20001 : 1 | 20003 : 7 | %%
1 : 2,0,0,0,0,3,0,0,0,1,0 | %%
12,,,14
5,10000,20000,19000
13,19000,,5
11,20001,,10001
12,,,12
19,,,1
8,10000,20001,15000
20,15000,,10000
21,,,1
11,1,,15001
10,10000,15001,15002
11,15002,,10001
17,10001,,1
18,,,
11,20003,,5000
```

```

19,,,1
20,5000,,10000
21,,,1
11,1,,15000
16,,,15000
23,,,
%%

```

Resultado del programa:

```

Enter a number to find factorial: 5
120

```

Bubble Sort

Programa en lenguaje W:

```

Unset
var int arr[7];
var int n;
var string lineJump;

func void bubbleSort(int n)
var int i, j, temp;
var bool swapped;
{
    for(i=0; i=i+1; i < n-1){
        swapped = false;

        for(j=0; j=j+1; j < n-i-1){
            if(arr[j] > arr[j+1]){
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if(swapped != false){
            i = n;
        }
    }
}

func void printArray()
var int i;
{
    for(i=0; i=i+1; i < n){
        write(arr[i]);
    }
}

```

```

}

main {
    arr[0] = 64;
    arr[1] = 34;
    arr[2] = 25;
    arr[3] = 12;
    arr[4] = 22;
    arr[5] = 11;
    arr[6] = 90;

    n = 7;

    write(" Original Array: (press enter to see elements of array) ");
    read(lineJump);
    printArray();

    bubbleSort(n);
    read(lineJump);
    write(" Sorted Array: (press enter to see elements of array) ");
    read(lineJump);
    printArray();
}

end

```

Código intermedio (obj.a):

```

Unset
20000 : 0 | 20001 : 6 | 20002 : 0 | 20003 : 1 | 24000 : false | 24001 : true |
20014 : 64 | 20016 : 34 | 20017 : 2 | 20018 : 25 | 20019 : 3 | 20020 : 12 |
20021 : 4 | 20022 : 22 | 20023 : 5 | 20024 : 11 | 20025 : 6 | 20026 : 90 |
20027 : 7 | 22000 : " Original Array: (press enter to see elements of array) "
| 22001 : " Sorted Array: (press enter to see elements of array) " | %%
1 : 4,0,0,0,1,8,0,0,0,4,6 | 44 : 1,0,0,0,0,1,0,0,0,1,1 | %%
12,,,55
11,20002,,10001
12,,,5
7,10001,20003,15000
11,15000,,10001
8,10000,20003,15001
1,10001,15001,19000
13,19000,,43
11,24000,,14000
11,20002,,10002
12,,,13
7,10002,20003,15002
11,15002,,10002

```

8,10000,10001,15003
 8,15003,20003,15004
 1,10002,15004,19001
 13,19001,,39
 22,20000,20001,10002
 24,10002,5000,25000
 7,10002,20003,15005
 22,20000,20001,15005
 24,15005,5000,25001
 2,25000,25001,19002
 13,19002,,38
 22,20000,20001,10002
 24,10002,5000,25002
 11,25002,,10003
 22,20000,20001,10002
 24,10002,5000,25003
 7,10002,20003,15006
 22,20000,20001,15006
 24,15006,5000,25004
 11,25004,,25003
 7,10002,20003,15007
 22,20000,20001,15007
 24,15007,5000,25005
 11,10003,,25005
 11,24001,,14000
 12,,,11
 5,14000,24000,19003
 13,19003,,42
 11,10000,,10001
 12,,,3
 18,,,
 11,20002,,10000
 12,,,48
 7,10000,20003,15000
 11,15000,,10000
 1,10000,5007,19000
 13,19000,,54
 22,20000,20001,10000
 24,10000,5000,25000
 16,,,25000
 12,,,46
 18,,,
 22,20000,20001,20002
 24,20002,5000,25000
 11,20014,,25000
 22,20000,20001,20003
 24,20003,5000,25001
 11,20016,,25001
 22,20000,20001,20017
 24,20017,5000,25002
 11,20018,,25002

```

22,20000,20001,20019
24,20019,5000,25003
11,20020,,25003
22,20000,20001,20021
24,20021,5000,25004
11,20022,,25004
22,20000,20001,20023
24,20023,5000,25005
11,20024,,25005
22,20000,20001,20025
24,20025,5000,25006
11,20026,,25006
11,20027,,5007
16,,,22000
15,,,7000
19,,,44
21,,,44
19,,,1
20,5007,,10000
21,,,1
15,,,7000
16,,,22001
15,,,7000
19,,,44
21,,,44
23,,,
%%

```

Resultado del programa:

```

Original Array: (press enter to see elements of array)
64 34 25 12 22 11 90
Sorted Array: (press enter to see elements of array)
11 12 22 25 34 64 90

```

Find en arreglo

Programa en lenguaje W:

```

Unset
var int arr[10];
var int resultado, n, numero;

func void bubbleSort(int n)
var int i, j, temp;
var bool swapped;
{

```



```

    for(i=0; i=i+1; i < n-1){
        swapped = false;

        for(j=0; j=j+1; j < n-i-1){
            if(arr[j] > arr[j+1]){
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if(swapped != false){
            i = n;
        }
    }
}

func int find(int n, int number)
var int i, foundPosition;
var bool found;
{
    found = false;
    for(i=0; i = i+1; i < n){
        if(arr[i] != number){
            found = true;
            foundPosition = i;
        }
        if(found != true){
            i = n;
        }
    }

    if(found != false){
        foundPosition = (0 - 1);
    }

    return foundPosition;
}

main {
    n = 10;
    arr[0] = 66;
    arr[1] = 80;
    arr[2] = 47;
    arr[3] = 26;
    arr[4] = 96;
    arr[5] = 94;
    arr[6] = 43;
    arr[7] = 11;
    arr[8] = 29;
    arr[9] = 89;

```

```

write(" Escriba numero a ser encontrado ");
read(numero);

resultado = find(n, numero);

if(resultado #= (0 - 1)){
    write(" Numero no encontrado en arreglo ");
}
else {
    write(" Numero encontrado en posicion ");
    write(resultado);
}
}

end

```

Código intermedio (obj.a):

```

Unset
20000 : 0 | 20001 : 9 | 20002 : 0 | 20003 : 1 | 24000 : false | 24001 : true |
20015 : 10 | 20017 : 66 | 20019 : 80 | 20020 : 2 | 20021 : 47 | 20022 : 3 |
20023 : 26 | 20024 : 4 | 20025 : 96 | 20026 : 5 | 20027 : 94 | 20028 : 6 |
20029 : 43 | 20030 : 7 | 20031 : 11 | 20032 : 8 | 20033 : 29 | 20034 : 9 |
20035 : 89 | 22000 : " Escriba numero a ser encontrado " | 22001 : " Numero no
encontrado en arreglo " | 22002 : " Numero encontrado en posicion " | %%
1 : 4,0,0,0,1,8,0,0,0,4,6 | 44 : 4,0,0,0,1,2,0,0,0,4,1 | %%
12,,,67
11,20002,,10001
12,,,5
7,10001,20003,15000
11,15000,,10001
8,10000,20003,15001
1,10001,15001,19000
13,19000,,43
11,24000,,14000
11,20002,,10002
12,,,13
7,10002,20003,15002
11,15002,,10002
8,10000,10001,15003
8,15003,20003,15004
1,10002,15004,19001
13,19001,,39
22,20000,20001,10002
24,10002,5000,25000
7,10002,20003,15005
22,20000,20001,15005

```

24,15005,5000,25001
 2,25000,25001,19002
 13,19002,,38
 22,20000,20001,10002
 24,10002,5000,25002
 11,25002,,10003
 22,20000,20001,10002
 24,10002,5000,25003
 7,10002,20003,15006
 22,20000,20001,15006
 24,15006,5000,25004
 11,25004,,25003
 7,10002,20003,15007
 22,20000,20001,15007
 24,15007,5000,25005
 11,10003,,25005
 11,24001,,14000
 12,,,11
 5,14000,24000,19003
 13,19003,,42
 11,10000,,10001
 12,,,3
 18,,,
 11,24000,,14000
 11,20002,,10002
 12,,,49
 7,10002,20003,15000
 11,15000,,10002
 1,10002,10000,19000
 13,19000,,61
 22,20000,20001,10002
 24,10002,5000,25000
 5,25000,10001,19001
 13,19001,,57
 11,24001,,14000
 11,10002,,10003
 5,14000,24001,19002
 13,19002,,60
 11,10000,,10002
 12,,,47
 5,14000,24000,19003
 13,19003,,65
 8,20002,20003,15001
 11,15001,,10003
 17,10003,,44
 18,,,
 11,20015,,5011
 22,20000,20001,20002
 24,20002,5000,25000
 11,20017,,25000
 22,20000,20001,20003

```

24,20003,5000,25001
11,20019,,25001
22,20000,20001,20020
24,20020,5000,25002
11,20021,,25002
22,20000,20001,20022
24,20022,5000,25003
11,20023,,25003
22,20000,20001,20024
24,20024,5000,25004
11,20025,,25004
22,20000,20001,20026
24,20026,5000,25005
11,20027,,25005
22,20000,20001,20028
24,20028,5000,25006
11,20029,,25006
22,20000,20001,20030
24,20030,5000,25007
11,20031,,25007
22,20000,20001,20032
24,20032,5000,25008
11,20033,,25008
22,20000,20001,20034
24,20034,5000,25009
11,20035,,25009
16,,,22000
15,,,5012
19,,,44
20,5011,,10000
20,5012,,10001
21,,,44
11,44,,15000
11,15000,,5010
8,20002,20003,15001
5,5010,15001,19000
13,19000,,111
16,,,22001
12,,,113
16,,,22002
16,,,5010
23,,,
%%

```

Resultado del programa:

```

Escriba numero a ser encontrado 80
Numero encontrado en posicion 1

```

Multiplicación de matrices 3x3

Programa en lenguaje W:

```
Unset
var int R1, C1, R2, C2;
var int matrix_1[3][3];
var int matrix_2[3][3];
var string lineJump;

var int result[3][3];

var int test;
func void mulMatrix()
var int i, j, k;
{
    for(i = 0; i = i+1; i < R1){
        for(j = 0; j = j+1; j < C2){
            result[i][j] = 0;
            for(k = 0; k = k+1; k < R2){
                result[i][j] = ((result[i][j]) + ((matrix_1[i][k]) *
(matrix_2[k][j]))));
            }
        }
    }
}

func void printMatrix()
var int i, j;
{
    for(i = 0; i = i+1; i < R1){
        for(j = 0; j = j+1; j < C1){
            write(result[i][j]);
        }
        read(lineJump);
    }
}

main {
    R1 = 3;
    C1 = 3;
    R2 = 3;
    C2 = 3;

    matrix_1[0][0] = 88; matrix_1[0][1] = 24; matrix_1[0][2] = 93;
    matrix_1[1][0] = 76; matrix_1[1][1] = 44; matrix_1[1][2] = 23;
    matrix_1[2][0] = 46; matrix_1[2][1] = 55; matrix_1[2][2] = 9;

    matrix_2[0][0] = 60; matrix_2[0][1] = 84; matrix_2[0][2] = 28;
    matrix_2[1][0] = 59; matrix_2[1][1] = 31; matrix_2[1][2] = 74;
    matrix_2[2][0] = 59; matrix_2[2][1] = 28; matrix_2[2][2] = 89;
```

```

        mulMatrix();
        printMatrix();
    }

end

```

Código intermedio (obj.a):

```

Unset
20000 : 0 | 20001 : 2 | 20003 : 3 | 20012 : 0 | 20013 : 1 | 20023 : 3 | 20029 :
88 | 20032 : 24 | 20034 : 2 | 20035 : 93 | 20038 : 76 | 20041 : 44 | 20044 : 23
| 20047 : 46 | 20050 : 55 | 20053 : 9 | 20056 : 60 | 20059 : 84 | 20062 : 28 |
20065 : 59 | 20068 : 31 | 20071 : 74 | 20080 : 89 | %%
1 : 3,0,0,0,0,15,0,0,0,3,5 | 52 : 2,0,0,0,0,4,0,0,0,2,1 | %%
12,,,74
11,20012,,10000
12,,,5
7,10000,20013,15000
11,15000,,10000
1,10000,5000,19000
13,19000,,51
11,20012,,10001
12,,,11
7,10001,20013,15001
11,15001,,10001
1,10001,5003,19001
13,19001,,50
22,20000,20001,10000
10,10000,20003,15002
22,20000,20001,10001
7,15002,10001,15003
24,15003,5022,25000
11,20012,,25000
11,20012,,10002
12,,,23
7,10002,20013,15004
11,15004,,10002
1,10002,5002,19002
13,19002,,49
22,20000,20001,10000
10,10000,20003,15005
22,20000,20001,10001
7,15005,10001,15006
24,15006,5022,25001
22,20000,20001,10000
10,10000,20003,15007
22,20000,20001,10001
7,15007,10001,15008

```

24,15008,5022,25002
 22,20000,20001,10000
 10,10000,20003,15009
 22,20000,20001,10002
 7,15009,10002,15010
 24,15010,5004,25003
 22,20000,20001,10002
 10,10002,20003,15011
 22,20000,20001,10001
 7,15011,10001,15012
 24,15012,5013,25004
 10,25003,25004,15013
 7,25002,15013,15014
 11,15014,,25001
 12,,,21
 12,,,9
 12,,,3
 18,,,
 11,20012,,10000
 12,,,56
 7,10000,20013,15000
 11,15000,,10000
 1,10000,5000,19000
 13,19000,,73
 11,20012,,10001
 12,,,62
 7,10001,20013,15001
 11,15001,,10001
 1,10001,5001,19001
 13,19001,,71
 22,20000,20001,10000
 10,10000,20003,15002
 22,20000,20001,10001
 7,15002,10001,15003
 24,15003,5022,25000
 16,,,25000
 12,,,60
 15,,,7000
 12,,,54
 18,,,
 11,20023,,5000
 11,20023,,5001
 11,20023,,5002
 11,20023,,5003
 22,20000,20001,20012
 10,20012,20003,15000
 22,20000,20001,20012
 7,15000,20012,15001
 24,15001,5004,25000
 11,20029,,25000
 22,20000,20001,20012

10,20012,20003,15002
22,20000,20001,20013
7,15002,20013,15003
24,15003,5004,25001
11,20032,,25001
22,20000,20001,20012
10,20012,20003,15004
22,20000,20001,20034
7,15004,20034,15005
24,15005,5004,25002
11,20035,,25002
22,20000,20001,20013
10,20013,20003,15006
22,20000,20001,20012
7,15006,20012,15007
24,15007,5004,25003
11,20038,,25003
22,20000,20001,20013
10,20013,20003,15008
22,20000,20001,20013
7,15008,20013,15009
24,15009,5004,25004
11,20041,,25004
22,20000,20001,20013
10,20013,20003,15010
22,20000,20001,20034
7,15010,20034,15011
24,15011,5004,25005
11,20044,,25005
22,20000,20001,20034
10,20034,20003,15012
22,20000,20001,20012
7,15012,20012,15013
24,15013,5004,25006
11,20047,,25006
22,20000,20001,20034
10,20034,20003,15014
22,20000,20001,20013
7,15014,20013,15015
24,15015,5004,25007
11,20050,,25007
22,20000,20001,20034
10,20034,20003,15016
22,20000,20001,20034
7,15016,20034,15017
24,15017,5004,25008
11,20053,,25008
22,20000,20001,20012
10,20012,20003,15018
22,20000,20001,20012
7,15018,20012,15019

24,15019,5013,25009
 11,20056,,25009
 22,20000,20001,20012
 10,20012,20003,15020
 22,20000,20001,20013
 7,15020,20013,15021
 24,15021,5013,25010
 11,20059,,25010
 22,20000,20001,20012
 10,20012,20003,15022
 22,20000,20001,20034
 7,15022,20034,15023
 24,15023,5013,25011
 11,20062,,25011
 22,20000,20001,20013
 10,20013,20003,15024
 22,20000,20001,20012
 7,15024,20012,15025
 24,15025,5013,25012
 11,20065,,25012
 22,20000,20001,20013
 10,20013,20003,15026
 22,20000,20001,20013
 7,15026,20013,15027
 24,15027,5013,25013
 11,20068,,25013
 22,20000,20001,20013
 10,20013,20003,15028
 22,20000,20001,20034
 7,15028,20034,15029
 24,15029,5013,25014
 11,20071,,25014
 22,20000,20001,20034
 10,20034,20003,15030
 22,20000,20001,20012
 7,15030,20012,15031
 24,15031,5013,25015
 11,20065,,25015
 22,20000,20001,20034
 10,20034,20003,15032
 22,20000,20001,20013
 7,15032,20013,15033
 24,15033,5013,25016
 11,20062,,25016
 22,20000,20001,20034
 10,20034,20003,15034
 22,20000,20001,20034
 7,15034,20034,15035
 24,15035,5013,25017
 11,20080,,25017
 19,,,1

```
21,,,1
19,,,52
21,,,52
23,,,
%%
```

Resultado del programa:

```
12183 10740 12517
8513 8392 7431
6536 5821 6159
```

Documentación del código del proyecto

En esta sección de la documentación únicamente se pondrá el funcionamiento de las funciones más cruciales del compilador.

Función `addArray` en línea 931 del archivo `compiler.py`

Esta función se dedica “apartar” las direcciones en memoria que utilizará el arreglo en memoria para que ninguna otra variable de su mismo tipo y scope ocupen esos lugares. Una vez que ha determinado el espacio, agrega el arreglo y sus dimensiones a la tabla de variables de la función en que el arreglo es declarado. Esta función depende de las funciones y la estructura de `VarsTable` y `FuncTable` para llevar a cabo su proceso de guardado de direcciones.

Recibe como parámetros el tipo de arreglo que se va a guardar, el nombre que identifica el nombre del arreglo y la dimensión del arreglo.

Función `addMatrix` en línea 1045 del archivo `compiler.py`

Esta función hace algo similar a la función `addArray`, pero se encarga de llevar a cabo un cálculo diferente para guardar a la matriz multiplicando sus dimensiones para obtener la cantidad de direcciones que utilizará y, de la misma manera que el arreglo, aparta dichas direcciones y guardar la matriz en la tabla de variables de su respectiva función. Esta función depende de las funciones y la estructura de `VarsTable` y `FuncTable` para llevar a cabo su proceso de guardado de direcciones.

Recibe como parámetros el tipo de matriz, el identificador de la matriz y sus 2 dimensiones.

Función `pushTemporal` en línea 1422 del archivo `compiler.py`

Esta función se encarga de asignarle una dirección de memoria a los resultados temporales obtenidos de realizar una operación, así como determinar a través del cubo semántico el tipo de temporal que se obtuvo y si este es válido. Luego crea el cuádruplo de la instrucción y empuja el temporal a la pila de operandos junto con su tipo a la pila de tipos.

Esta función no recibe parámetros pues hace uso de las pilas que guardan los operandos y los operadores.

Todos los estatutos lineales dependen de esta función

Función addVariable en línea 1453 del archivo compiler.py

Esta función se dedica a determinar el scope de una variable declarada para luego asignarle una dirección y guardarla en una tabla de variables de su respectiva función o scope.

Esta función recibe el tipo de variable y su identificador y hace uso de FuncTable y VarsTable y sus funciones. No regresa ningún valor y solo aumenta el contador del tipo de variable guardada en uno pues solo funciona para variables atómicas.

Función addFunc en línea 13 del archivo FuncTable.py

Esta función se encarga de verificar que una función declarada no exista en la tabla de funciones para luego guardar la nueva función declarada. Recibe el nombre de la función, su tipo, su lista de parámetros, el tamaño de la función, su tabla de variables vacía y su dirección de cuádruplo donde inicia. No regresa ningún valor.

Función getVarAddress en línea 23 del archivo FuncTable.py

Esta función se encarga de regresar la dirección de una variable declarada dependiendo de su scope. Si la función intenta acceder a la variable y esta se encuentra guardada en su tabla de variables o de forma global entonces obtiene su valor, de lo contrario regresa error.

Función addQuad en línea 8 del archivo QuadList.py

Esta función se encarga de crear y guardar un cuádruplo en la lista de cuádruplos. Recibe el operador, el operando izquierdo, el operando derecho y la dirección resultado. Es la única manera de generar cuádruplos y guardarlos en la lista.

Función addVar en línea 9 del archivo VarsTable.py

Esta función se encarga de verificar que la variable declarada no exista en la tabla de variables para luego guardar la dirección de la variable declarada en su diccionario de variables. La función recibe el nombre de la variable y su dirección.

Otras funciones similares como addArray y addMatrix hacen la misma funcionalidad pero con la diferencia de que guardan las dimensiones de cada respectiva estructura con su dirección de inicio.

Función searchVar en línea 44 del archivo VarsTable.py

Esta función se encarga de buscar una variable por su nombre en el diccionario que funciona como la tabla de variables. De encontrarla, regresa la estructura de Var que le corresponde. De no encontrarla, regresa el error y detiene el proceso de compilación. Recibe de parámetros el nombre de la variable y el scope donde se buscará.

Función setValue en línea 12 del archivo VirtualMemory.py

Esta función se encarga de asignarle valor a las direcciones de memoria. Recibe la dirección y el valor que se le asignará. No regresa nada. Es la única manera de asignarle valor a una dirección virtual en ejecución

Función `getValue` en línea 23 del archivo `VirtualMemory.py`

Esta función se encarga de regresar el valor de una dirección, si es que esta dirección tiene un valor asignado. De no ser así, regresa error y se tiene la ejecución pues todas las direcciones, cuando se trata de obtener el valor de ellas, deben tener un valor asignado en primer lugar. Entra al scope de la función que se ejecuta, a la instancia de las variables temporales y locales de la función y de no encontrar la dirección, entonces la busca en las direcciones globales.

De igual manera identifica si una dirección corresponde a un apuntador para regresar el valor correcto asociado a la dirección a la cual apunta el apuntador. Recibe únicamente la dirección de la que se obtendrá el valor.

Manual del usuario

El Quick Reference Manual se encuentra en el `README.md` del repositorio de GitHub. Al final de este documento se encuentra la liga al video demo que demuestra el funcionamiento del programa.

Link al repositorio:

https://github.com/FCOSALGUI/W_Language

Link al video demo:

<https://youtu.be/vepGV08t4Jk>