



Tecnológico de Monterrey

Diseño de Compiladores

3 de Junio del 2021

Documentación:

Basic ++

Integrantes:

David Martínez Valdés A00820087

Ulises Serrano Martínez A01233000

Índice

1. Descripción del Proyecto	3
1.1 Propósito y Alcance del Proyecto	3
1.2 Análisis de Requerimientos y Test Cases	3
1.3 Descripción del Proceso	4
2. Descripción del Lenguaje	6
2.1 Nombre del lenguaje	6
2.2 Descripción genérica de las principales características del lenguaje	6
2.3 Listado de errores que pueden ocurrir	6
3. Descripción del Compilador	8
3.1 Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.	8
3.2 Descripción del análisis Léxico	8
3.2.1 Patrones de construcción	8
3.2.2 Enumeración de Tokens	8
3.3 Descripción del análisis de sintaxis	9
3.3.1 Gramática Formal	9
3.4 Descripción de generación de código Intermedio y Análisis Semántico	13
3.4.1 Código de operación y direcciones virtuales asociadas a los elementos del código.	13
3.4.2 Diagramas de Sintaxis	14
3.4.3 Descripción de cada una de las acciones semánticas y de generación de código	21
3.4.4 Tabla de consideraciones semánticas	23
3.5 Descripción del proceso de Administración de memoria usado en la Compilación	23
3.5.1 Especificación gráfica de cada estructura de datos	23
4. Descripción de la Máquina Virtual	26
4.1 Equipo de computo, lenguaje y utilerías especiales usadas.	26
4.2 Descripción detallada del proceso de administración de memoria en ejecución	26
4.2.1 Especificación gráfica de cada estructura de datos	26
4.2.2 Asociación hecha entre las direcciones virtuales (compliación) y las reales (ejecución)	27
5 Pruebas del funcionamiento del lenguaje	28
5.1 Pruebas de funcionamiento del proyecto	28
5.1.1 Codificación de la prueba	28
5.1.2 Resultados arrojados por la generación de código intermedio y por la ejecución.	32
6. Documentación del código del proyecto	38

1. Descripción del Proyecto

1.1 Propósito y Alcance del Proyecto

El propósito de este proyecto es facilitar el aprendizaje de manera que se pueda practicar conceptos básicos de programación desde el celular.

El alcance es crear un lenguaje que permita las acciones básicas de un lenguaje de programación, como lo son ciclos y condicionales. Además, se requiere que se pueda correr en un celular.

1.2 Análisis de Requerimientos y Test Cases

Las acciones a desarrollar son las siguientes:

- Declaración de variables globales y locales a una función de tipo entero, flotante y char. Estas pueden ser dimensionadas con una o dos dimensiones.
- Declaración de funciones de tipo entero, flotante, char o void. Los parámetros siguen la sintaxis de la declaración de variables de tipo simple y únicamente son de entrada.
- Uso de estatutos:
 - **Asignación:** A un identificador (o a una casilla o a un atributo) se le asigna el valor de una expresión o el valor que regresa una función.
 - **Llamada a función void:** Se manda llamar una función o método que no regresa valor (caso de funciones void).
 - **Retorno de una función:** Este estatuto va dentro de las funciones e indica el valor de retorno (si no es void).
 - **Lectura:** Se puede leer uno o más identificadores (o una casilla) separados por comas.
 - **Escritura:** Se pueden escribir letreros y/o resultados de expresiones separadas por comas.
 - **Estatutos de decisión:** Evalúa una expresión y dependiendo del resultado ejecuta el bloque de código. Puede incluir un bloque de código a ejecutar solamente cuando no se cumple la expresión.
 - **Estatutos de repetición:**
 - Condicional: Repite los estatutos mientras la expresión sea verdadera.
 - No condicional: Repite hasta que no se cumpla la expresión, brincando de 1 en 1.
 - **Expresiones:** Las expresiones son las tradicionales. Existen los operadores aritméticos, lógicos y relacionales: +, -, *, /, %, && (and), || (or), <, <=, >, >=, ==. Se manejan las prioridades tradicionales, se pueden emplear paréntesis para alterarla. Existen identificadores, palabras

reservadas, constantes enteras, constantes flotantes, constantes char y constantes string (letreros).

Con estas acciones se deberá poder cumplir con los siguientes escenarios de prueba:

- Cálculo de Factorial y Serie de Fibonacci (en versión cíclica y versión recursiva). (Con estas pruebas validamos: expresiones, entrada/salida, ciclos, decisiones, funciones, parámetros, etc.)
- Un Sort y un Find para un Vector. (Con estas pruebas validamos trabajo sobre elementos estructurados)
- Una multiplicación de matrices.

1.3 Descripción del Proceso

El proceso de desarrollo para la elaboración de este proyecto fue llevado a cabo por medio de sesiones semanales de aproximadamente 3 a 4 horas donde se hacían los avances correspondientes a esa semana.

Para el control de versiones se utilizó la herramienta de Git por medio de Github, de tal manera que a través de los commits se comentaba el avance realizado en la sesión. Además, se añadió un archivo README.md donde se fue registrando el avance de cada semana. Ver la liga para más detalle: <https://github.com/UlisesSerrano/Basic->

Primer Avance

Se implementó la gramática del lenguaje utilizando PLY para el lexer y el parser.

Segundo Avance

Se crearon las estructuras para el directorio de funciones y las tablas de variables. Se implementó el cubo semántico.

Tercer Avance

Se generó el código para las expresiones y los estatutos lineales.

Cuarto Avance

Se generó el código para los estatutos no lineales (condiciones y ciclos)

Quinto Avance

Se generó el código de funciones, quedó pendiente la lectura correcta de las expresiones de los parámetros y regresar el valor de la función cuando no es void.

Sexto Avance

Se corrigió el código de funciones y se comenzó con la implementación de la máquina virtual para estatutos lineales.

Séptimo Avance

Se implementó la generación de código de arreglos. Queda pendiente la máquina virtual.

Avance Final

Se corrigieron detalles de la generación de código en general. Se terminó la máquina virtual.

Por otra parte, para la estructura de los commits se siguió el siguiente formato:

feat(módulo): descripción de lo que se trabajó

Siendo “feat” una de las distintas categorías para describir en que se trabajó y “módulo” la sección donde se trabajó. Si se trabajó en distintas secciones se utilizó la palabra “app” para describir que se hicieron modificaciones a varios módulos.

Las posibles categorías de descripción son las siguientes:

- *feat*: Se desarrolló una nueva “feature” o funcionalidad.
- *fix*: Se hizo un “bug fix” para evitar un defecto
- *refactor*: Se hicieron cambios en el código que no afectan a la funcionalidad de la aplicación
- *docs*: Se agregó o modificó la documentación

Un ejemplo puede ser:

feat(parser_lexer): Add functions directory & var table code

Aprendizajes

David Martínez: Con este proyecto “integrador” fue posible darme cuenta de cómo las distintas habilidades que adquirí a lo largo de la carrera, ya sea por medio de las clases o experiencia profesional, son de gran utilidad para hacer un buen desarrollo utilizando buenas prácticas. Además, al integrar distintos conocimientos me fue posible fortalecer mis competencias tanto como estudiante como desarrollador, de manera que esta experiencia sirvió para confirmar mis conocimientos.

Ulises Serrano: Con este proyecto me he dado cuenta que hay varias tecnologías a la mano de las cuales uno puede hacer uso para complementar las herramientas que fue conociendo durante la carrera. También uno de los retos que se me presentaron en el desarrollo de este proyecto fue que en medio de la pandemia era un poco difícil juntarse físicamente, como uno ya está acostumbrado o estaba acostumbrado a la hora de tratar con los compañeros de su equipo.

2. Descripción del Lenguaje

2.1 Nombre del lenguaje

Basic ++

2.2 Descripción genérica de las principales características del lenguaje

El lenguaje cuenta con una estructura similar a C++ pero con algunas restricciones.

La declaración de variables se debe realizar al inicio del programa en el caso de las globales y al inicio de cada función en el caso de las locales.

Las llamadas a una función deben empezar con un ampersand (&) y luego seguir con el nombre de la función.

El programa debe contener el nombre del programa y la función main como mínimo para su ejecución.

Los arreglos y matrices deben tener números enteros positivos para el tamaño de sus dimensiones.

2.3 Listado de errores que pueden ocurrir

Compilación:

- Function already defined: Ocurre cuando se intenta declarar una función con el mismo nombre dos veces.
- Variable already defined: Ocurre cuando se intenta declarar una variable con el mismo nombre y en el mismo contexto (global o local) dos veces.
- Undeclared function: Si se trata de llamar a una función que no ha sido definida previamente.
- Type mismatch: Cuando se trata de compilar una operación de tipos no compatibles.
- Return type mismatch: Cuando el tipo de retorno de la expresión es distinto al de la función.
- Variable {id} has not dimensions: Cuando se intenta indexar una variable no dimensionada.
- Type mismatch on argument on call function: Al llamar una función, si alguno de sus argumentos no concuerda con el tipo de parámetro correspondiente arroja este error.
- Incorrect number of arguments: Cuando se trata de llamar una función con más argumentos que la cantidad de parámetros que tiene.
- Missing arguments: Cuando no se agregaron suficientes argumentos a una llamada de una función.
- Cannot call void function on expresion: Si dentro de una expresión se llama a una función void regresa este error.
- Syntax error on the Input: Muestra cualquier error de sintaxis.
- Illegal character: Muestra cualquier carácter que no sea parte de la gramática.

Ejecución:

- Index out of bounds: Si el resultado de la expresión de indexamiento no está dentro del rango permitido por la variable dimensionada.
- Invalid input type: Al ingresar un input si el tipo no es del tipo a la variable a asignar
- Wrong instruction: Si el archivo de cuádruplos contiene alguna instrucción no soportada por la máquina virtual.
- Address not found: Cuando se intenta acceder a un registro de memoria que no tiene valor.

3. Descripción del Compilador

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

Como equipo de compu utilizamos nuestras computadoras personales (Laptops Mac)

Todo el proyecto se desarrollo en el lenguaje de programación **Python**.

Como editor de código fuente usamos Visual Studio Code

Dentro del proyecto, se utilizaron distintas herramientas para el desarrollo de la aplicación:

- PLY: Analizador Léxico y sintáctico similar a Lex y Yacc

Además, se crearon utilerías nuevas para facilitar el desarrollo:

- Cubo semántico
- Stack

3.2 Descripción del análisis Léxico

3.2.1 Patrones de construcción

ID (identificador):	[a-zA-Z][a-zA-Z_\d]*
CTE_F (constante flotante):	\d*\.\d+
CTE_NEG_F (constante flotante negativa):	-\d*\.\d+
CTE_I (constante entera):	\d+
CTE_NEG_I (constante entera negativa):	-\d+
CTE_STRING (constante string):	\"([^\n] (\.))*?\"
CTE_CHAR (constante carácter):	(\[^\']*')

3.2.2 Enumeración de Tokens

Palabras reservadas:

- 'program'
- 'if'
- 'else'
- 'var'
- 'int'
- 'float'
- 'char'
- 'print'
- 'read'
- 'void'
- 'func'
- 'main'
- 'return'
- 'do'
- 'while'
- 'for'
- 'to'

Tokens:

- MINUS: '-'
- PLUS: '+'
- MULT: '*'
- DIV: '/'
- MOD: '%'
- SEMICOLON: ';'
- L_P: '('
- R_P: ')'
- COMA: ','
- L_B: '{'
- R_B: '}'
- L_SB: '['
- R_SB: ']'
- AND: '&&'
- OR: '||'
- EQ: '=='
- GREATERTHANEQ: '>='
- LESSTHANEQ: '<='
- GREATERTHAN: '>'
- LESSTHAN: '<'
- DIFERENT: '!='
- EQUAL: '=='
- AMP: '&'

3.3 Descripción del análisis de sintaxis

3.3.1 Gramática Formal

1. Programa:

program : PROGRAM ID main_quad SEMICOLON g_var func main

2. Main:

main : MAIN L_P params R_P var_declaration L_B main_start statements R_B

3. Tipo:

type : INT
| FLOAT
| CHAR'

4. Variables globales:

g_var : var_declaration

5. Funciones:

funcs : function funcs
| empty

6. Declaracion de variable:

var_declaration : VAR var1
| empty

7. var1:

var1 : var_type dec_id var2 SEMICOLON var4

8. var2:

var2 : COMA dec_id var3
| empty

9. var3:

var3 : var2

10. var4:

var4 : var1
| empty'

11. declaración de id:

dec_id : ID add_id dec_id1

12. declaración de id1:

dec_id1 : L_SB CTE_I R_SB dec_id2
| empty

13. declaración de id2:

dec_id2 : L_SB CTE_I R_SB
| empty'

14. id:

id : ID id1

15. id1:

id1 : L_SB expression R_SB id2
| empty

16.id2:

id2 : L_SB expression R_SB
| empty

17. Tipo de variable:

var_type : type

18. Tipo de función:

func_type : VOID
| type

19. Definición de parámetros:

params : var_type dec_id params1
| empty

20. Parámetros 1

params1 : COMA params
| empty'

21. Estatutos:

statements : statement statements
| empty

22. Estatuto:

statement : assignation
| call_func

- | return_func
- | read
- | write
- | decision_statement
- | repetition_statement
- | expression

23 Argumentos:

- args : args1
- | empty

24. Argumentos 1:

- args1 : expression args2

25 Argumentos 2:

- args2 : COMA args1
- | empty

26 Leer argumentos

- read_args : read_args1

27 Leer argumentos 1:

- read_args1 : COMA expression read_args1
- | empty

28 Escribir:

- write : PRINT L_P write_args R_P SEMICOLON

29. Regresar Función:

- return_func : RETURN L_P expression R_P SEMICOLON

30. Leer:

- read : READ L_P read_args R_P SEMICOLON

31. Argumentos de escritura

- write_args : write_args2 write_args1

32 Argumentos de escritura 1:

- write_args1 : COMA write_args2 write_args1
- | empty

33. Argumentos de escritura 2:

- write_args2 : expression
- | CTE_STRING add_cte_string

34. Llamada a función:

- call_func : AMP ID L_P args R_P SEMICOLON

35. Estatuto de repetición:

- repetition_statement : while_statement
- | for_statement

36. Operador 1:

- op1 : OR expression
- | empty

37. Operador 2:

op2 : AND texp
| empty

38. Operador 3:

op3 : LESSTHAN
| LESSTHANEQ
| GREATERTHAN
| GREATERTHANEQ
| EQ
| DIFERENT

39. Operador 4:

op4 : PLUS
| MINUS'

40. Operador 5:

op5 : MULT
| DIV
| MOD'

41. hacer declaraciones:

do_statement : DO L_B statements R_B

42. Expresión

expression : texp generate_quad_1 op1

43. T Expresión

texp : gexp generate_quad_2 op2

44. G Expresión:

gexp : mexp generate_quad_3 op3aux

45. M Expression

mexp : term generate_quad_4 op4aux

45. Término:

term : fact generate_quad_5 op5aux

46. Fact:

fact : call_func_exp
| id
| L_P expression R_P
| cte'

46 CTE:

cte : CTE_CHAR
| CTE_F
| CTE_NEG_F
| CTE_I
| CTE_NEG_I

47 OP1:

op1 : OR expression

| empty

48 OP2:

op2 : AND add_operator texp
| empty

49 OP3:

op3 : LESSTHAN
| LESSTHANEQ
| GREATERTHAN
| GREATERTHANEQ
| EQ
| DIFERENT

50 op3 AUX

op3aux : op3 gexp
| empty

51 op4:

op4 : PLUS
| MINUS

52 op4_aux:

op4aux : op4 mexp
| empty

53 op5:

op5 : MULT
| DIV
| MOD

53: op5 aux:

op5 : MULT
| DIV
| MOD

54 EMPTY:

empty :

3.4 Descripción de generación de código Intermedio y Análisis Semántico

3.4.1 Código de operación y direcciones virtuales asociadas a los elementos del código.

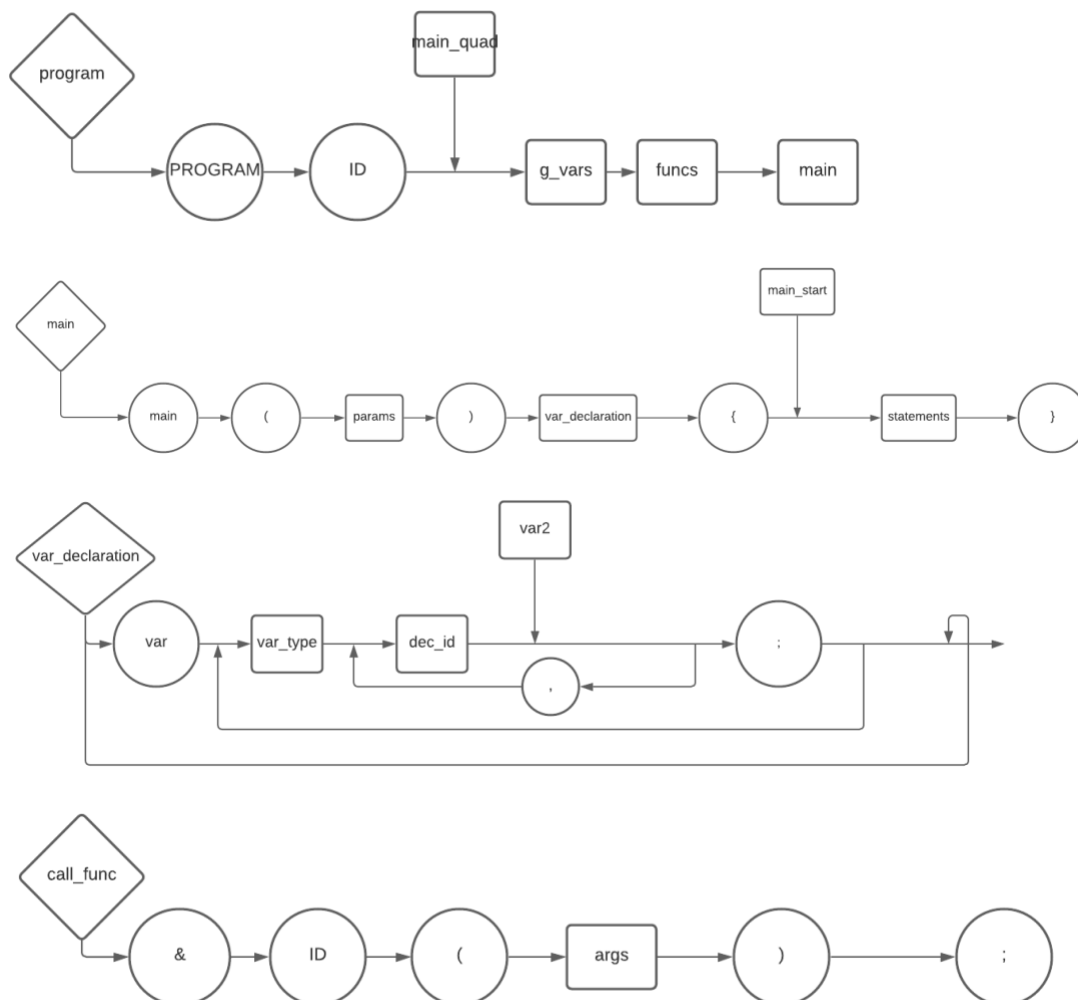
EL listado de códigos soportados por la máquina virtual es el siguiente:

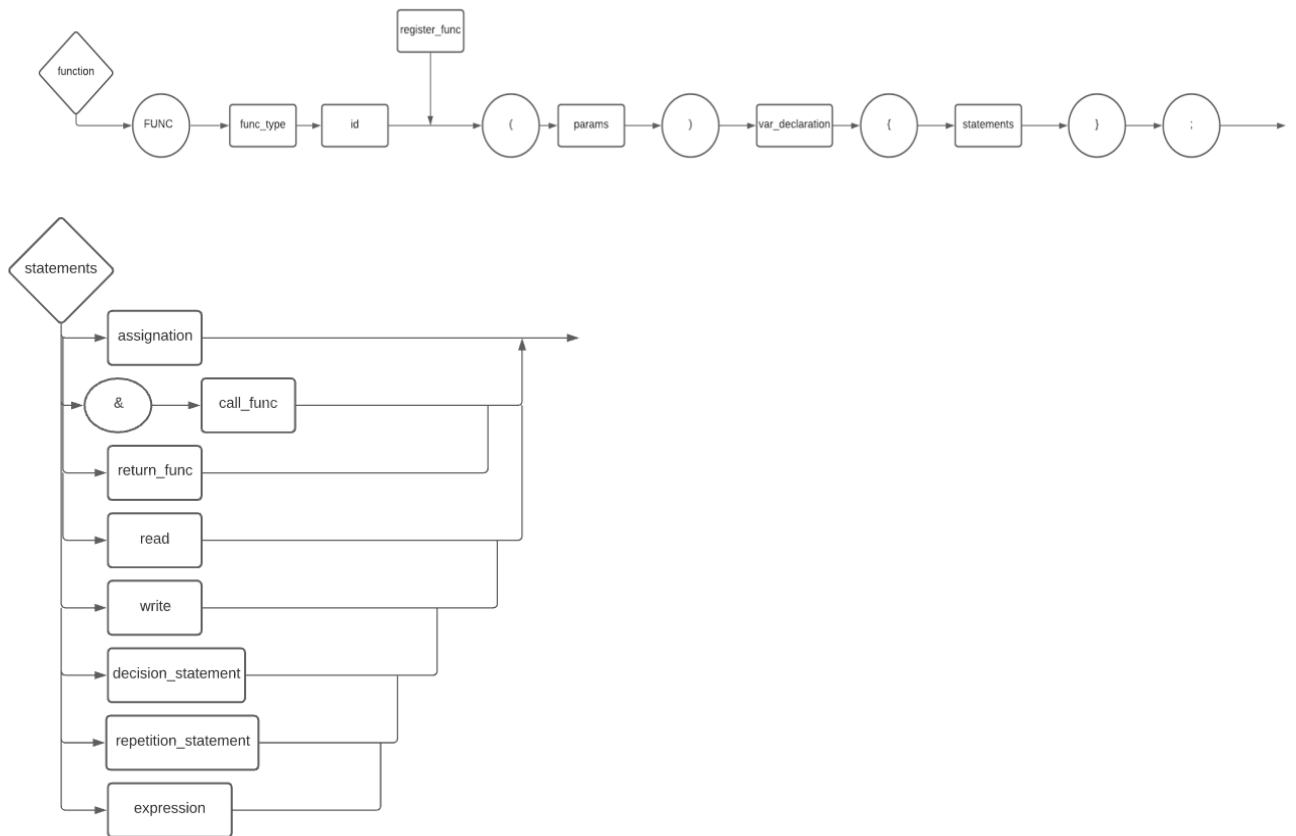
- goto
- gotoF
- goSub
- param
- ERA
- ENDFunc

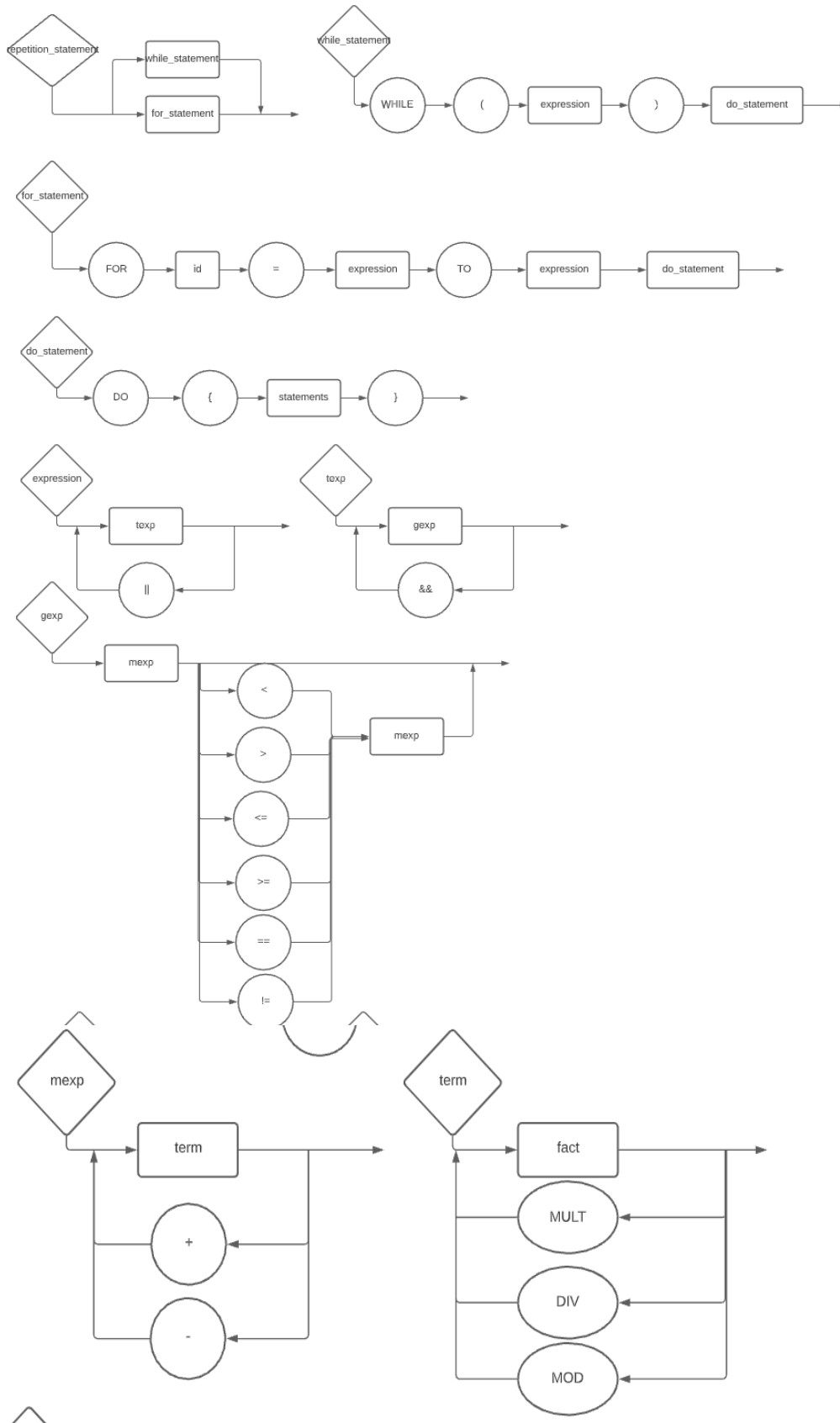
- print
- read
- return
- ver
- =
- +
- -
- *
- /
- %
- <
- >
- <=
- >=
- !=
- ==
- &&
- ||

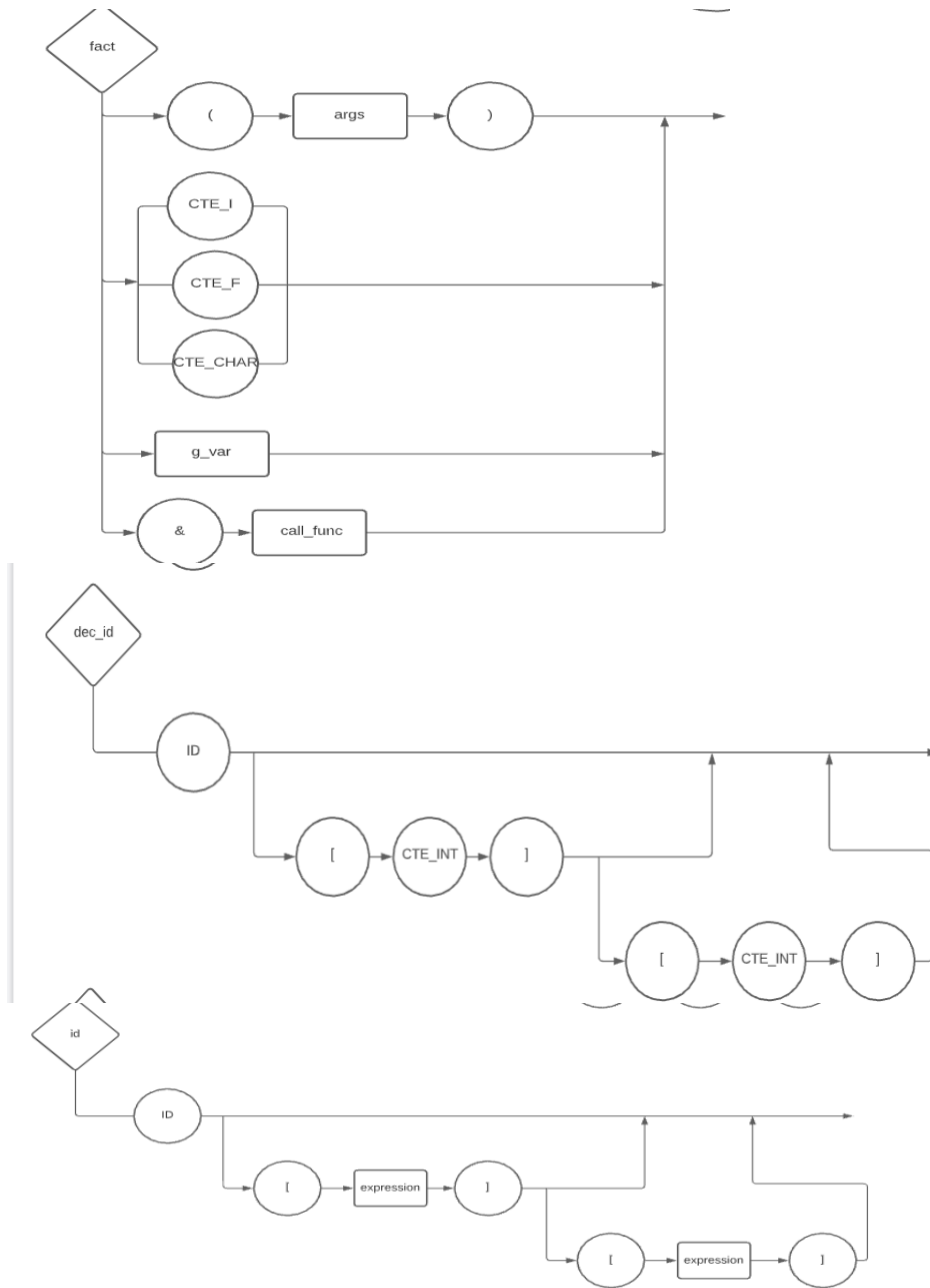
Los códigos se mantuvieron como strings por simplicidad.

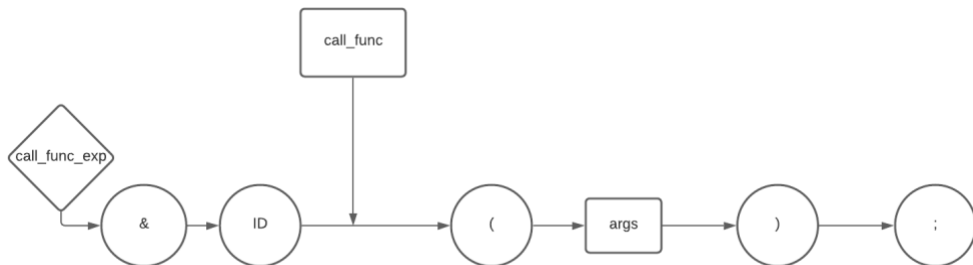
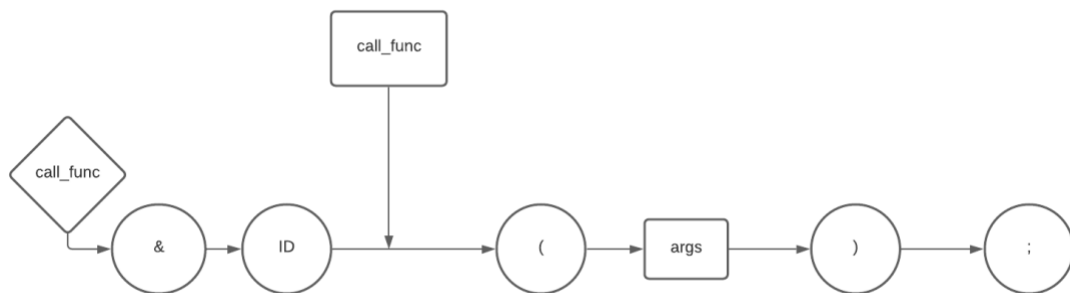
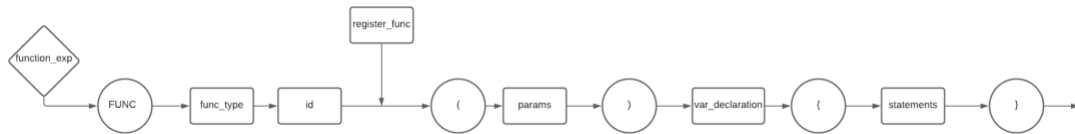
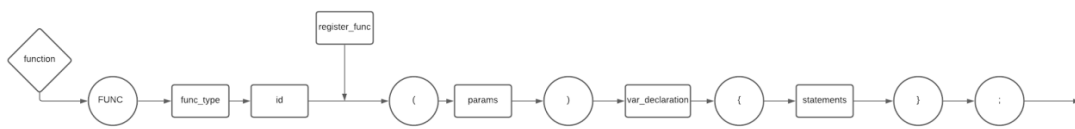
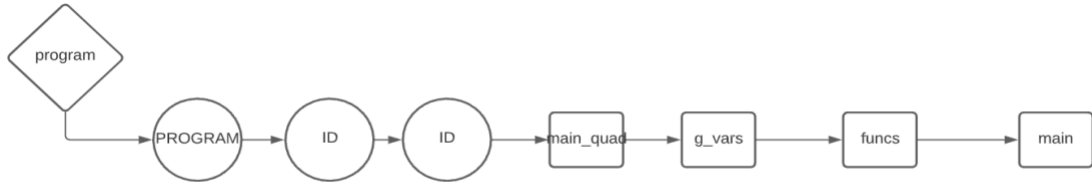
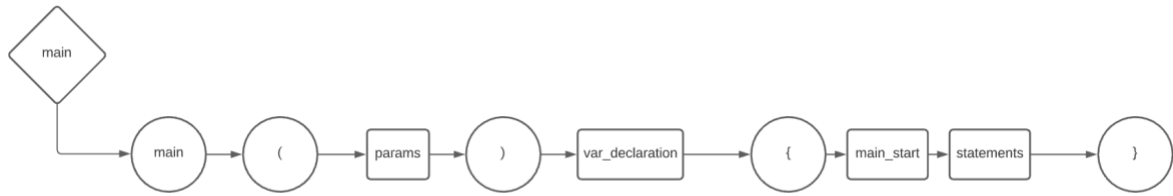
3.4.2 Diagramas de Sintaxis

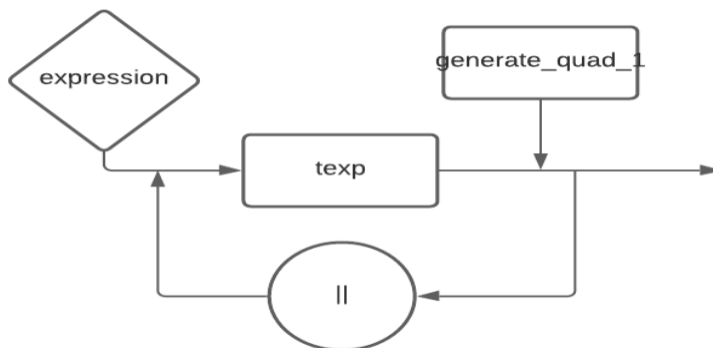
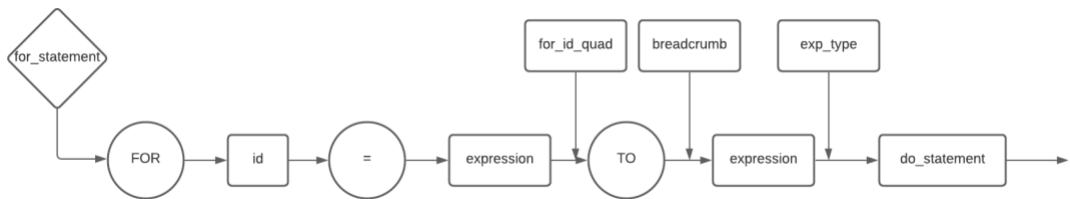
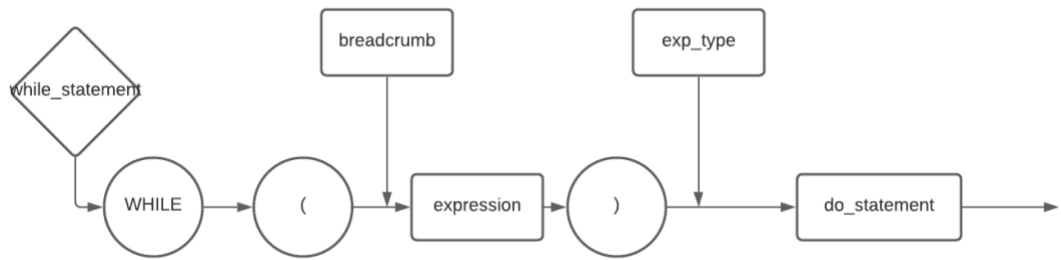
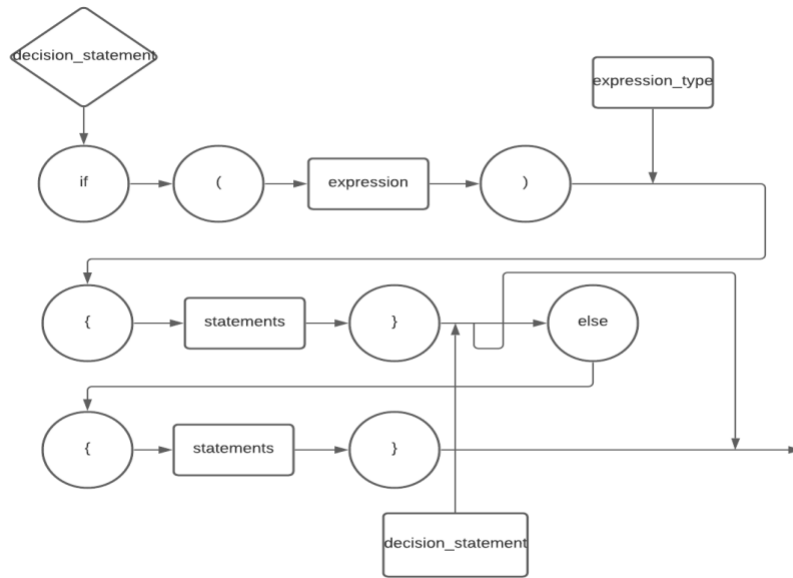


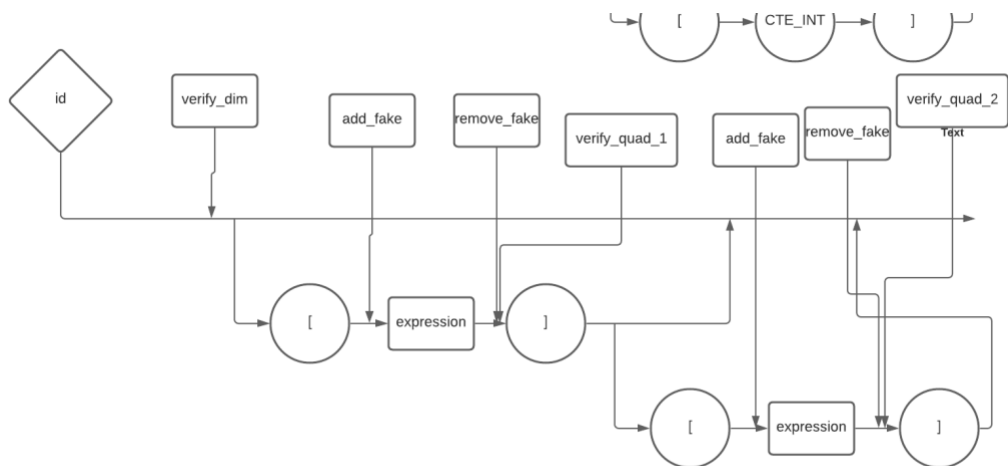
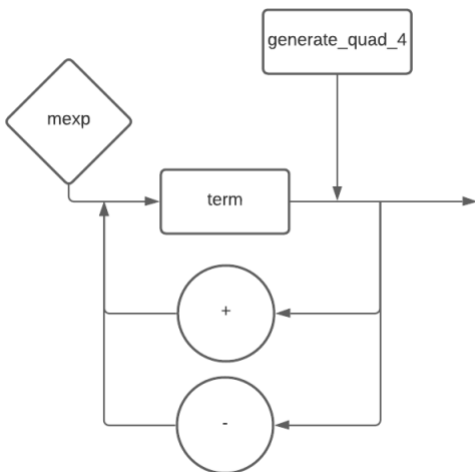
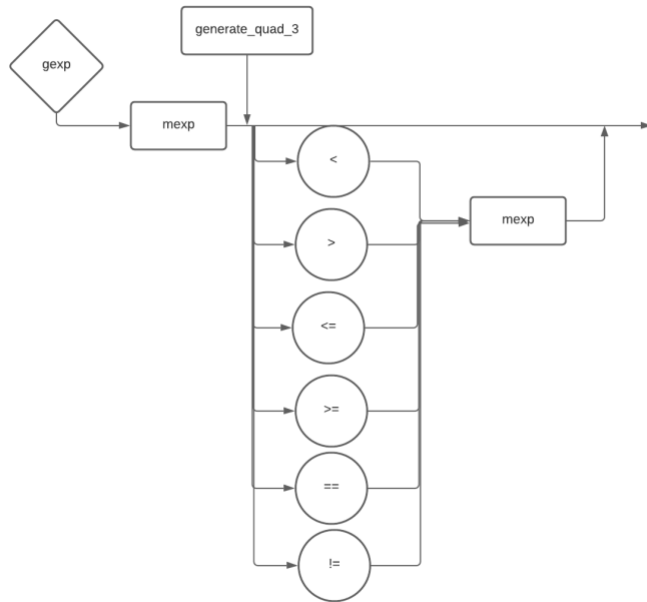


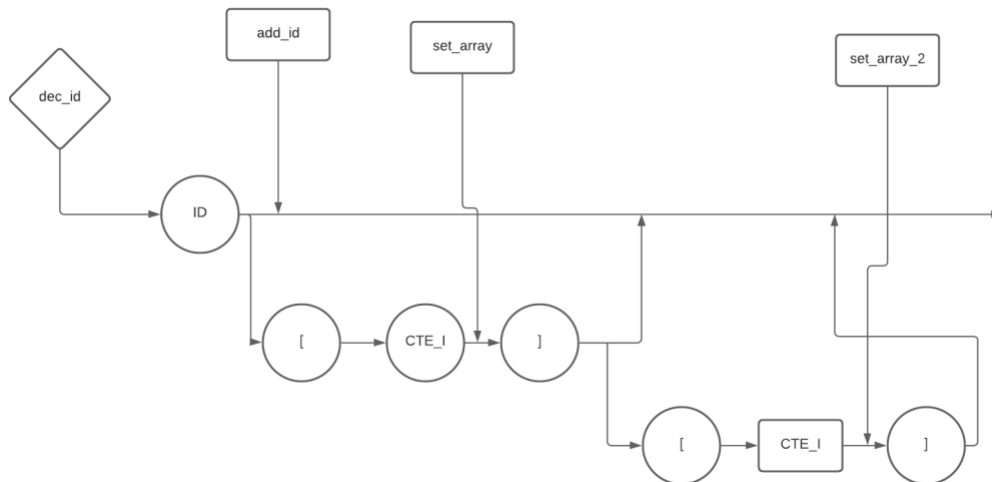












Ver liga para más detalle: https://lucid.app/lucidchart/8603e3dd-28cd-4c65-9c02-9195b544b21c/edit?beaconFlowId=C41A3B7F40621342&page=0_0#

3.4.3 Descripción de cada una de las acciones semánticas y de generación de código

generate_quad(): Función que genera el cuádruplo de la expresión a evaluar

fill(jump, counter): Función que sirve para rellenar los cuádruplos para el goto y el gotoF

p_main Agrega el main al directorio de funciones

main_start: Rellenar el goTo al main indicando donde inicia su ejecución

p_main_quad: Agregar el goTo al main como primer cuádruplo

p_add_id: Función que agrega el id a la tabla variable ya sea local o global pero valida que no exista antes de hacerlo

p_set_array: coloca la primera dimensión del id y ese número a la tabla de constante

p_set_array_2: coloca la segunda dimensión del id en la tabla de variable y ese número a la tabla de constantes

p_set_id: coloca el id que está leyendo

p_verify_dim: Verifica que la variable cuente con dimensiones

p_verify_quad_1: Agrega la verificación del cuádruplo y las operaciones para obtener el valor en memoria de acuerdo con la dimensión de la primera dimensión.

p_verify_quad_2: Agrega la verificación del cuádruplo y las operaciones para obtener el valor en memoria de acuerdo con la dimensión de la segunda dimensión.

p_add_base: Agrega la dirección base al apuntador the variable en memoria

p_function : Agrega atributos de función a la función en el directorio de funciones

p_register_func: Agrega la función al directorio de funciones

p_assignment: Asignación de cuádruplos.

p_param_check: Verifica si los argumentos coinciden con los parámetros de la función

p_call_func: llama a la función vacía con gosub

p_call_func_exp: Llama a la función dentro de la expresión con goSub

p_call_func_era: Agrega el cuádruplo ERA

p_return_func: Regresa la declaración con el cuádruplo

p_read: Agrega un cuádruplo para leer.

p_write_args2: Agrega un cuádruplo para Escribir

p_decision_statement: Llena el cuádruplo para saltar en caso de que haya un False

p_exp_type: Verifica que la expresión pueda ser evaluada y agrega el goto False

p_else_jump: Agrega el goto en caso de que la expresión es verdadera

p_for_statement: Se le agrega en 1 al id para la siguiente iteración

p_for_id: Coloca el id del for

p_for_id_quad: Asigna el valor a el id del for

p_breadcrumb: Guarda el contador para evaluar la extensión del ciclo

p_while_statement: Coloca el goto para checar nuevamente la expresión del while o para terminar el ciclo

p_generate_quad_1: Genera cuádruplos por cada tipo de expresión

p_add_fake: Agrega un fondo falso

p_remove_fake: Quita el fondo falso

p_id_quad: Agrega a las pilas los valores de cada expresión generada por el cuádruplo

p_add_cte_<tipo>: Agrega constantes a la tabla de constantes

p_add_operator: Agrega el operador a la pila para la expresión generada por el cuádruplo

3.4.4 Tabla de consideraciones semánticas

Oper1	Oper2	Operador													
		+	-	*	/	<	>	<=	>=	==	!=	&&		%	=
int	int	int	int	int	int	int	int	int	int	int	int	int	int	int	int
int	float	float	float	float	float	int	int	int	int	int	int	err	err	float	float
int	char	err	err	err	err	err	err	err	err	err	err	err	err	err	err
float	int	float	float	float	float	int	int	int	int	int	int	err	err	err	float
float	float	float	float	float	float	int	int	int	int	int	int	err	err	err	float
float	char	err	err	err	err	err	err	err	err	err	err	err	err	err	err
char	int	err	err	err	err	err	err	err	err	err	err	err	err	err	err
char	float	err	err	err	err	err	err	err	err	err	err	err	err	err	err
char	char	err	err	err	err	err	err	err	err	int	int	err	err	err	char

3.5 Descripción del proceso de Administración de memoria usado en la Compilación

3.5.1 Especificación gráfica de cada estructura de datos

Directorio de funciones

Se utiliza como llave y se almacena el nombre en “name” y tipo de función en “type”, además se le asigna una dirección de memoria global en “memory_address” para almacenar su valor de retorno en caso de regresar alguno. Continúa almacenando un arreglo que indica el orden y el tipo de los parámetros necesarios para hacer la llamada a la función en “param_types”. El siguiente elemento es el número que indica en qué cuádruplo de la lista comienza la ejecución de la función y se almacena en “start_quad”. Finalmente almacena en “variable_counter” un objeto que indica la cantidad de variables locales y temporales que se utilizan en la ejecución de la función

<u>name</u>	<u>type</u>	<u>memory_address</u>	<u>param_types</u>	<u>start_quad</u>	<u>variable_counter</u>
string	string	número	string[]	número	{ local: { int: número, float: número, char: número }, temp: { int: número, float: número,

					char: número } }
...

Tablas de variables

Se guarda el nombre de variable como la llave y en el primer elemento de un arreglo. En el siguiente elemento se almacena el tipo, seguido de la dirección de memoria según el rango permitido por su tipo. Finalmente en el último elemento se guarda otro arreglo que contiene las dimensiones de la variable si es que llega a tener.

<u>id</u>	type	address	dims
número	string	número	número[]
...

global_var_table: Para manejo de variables globales.

local_var_table: Para el manejo de variables locales a una función. Se reutilizó la misma variable ya que esta estructura se limpia al terminar de compilar una función.

Lista de cuádruplos

[

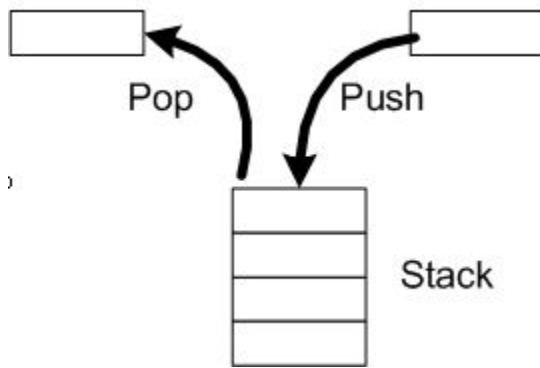
[operador: string, dir1: número, dir2: número, dir3: número]

...

]

Para el manejo de cuádruplos se utilizó una lista de arreglos, ya que los “tuplos” en Python no son modificables, de manera que no era posible editar elementos para ciertas operaciones que así lo necesitaban. El primer elemento almacena el operador que va a efectuar la máquina virtual. Los siguientes tres elementos son direcciones de memoria que varían dependiendo de la operación.

Manejo de Stacks



Para esto se creó una clase capaz de soportar las operaciones básicas sobre una lista que se maneja como LIFO (Last In - First Out), en la cual se permite agregar elementos al final de la lista (`push`), sacar el último elemento que entró (`pop`) y ver ese último elemento sin sacarlo de la lista (`peek`). Para ayuda adicional, a esta clase se le agregaron las opciones de ver si la lista está vacía (`is_empty`) y contar la cantidad de elementos (`size`).

Stacks para manejo de expresiones

`elements_stack`: Manejo de operandos u otros elementos como pointers que se utilicen para las expresiones

`types_stack`: Manejo de tipos de los elementos, contiene en el mismo orden los tipos de cada elemento de `elements_stack`

`operators_stack`: Manejo de operadores para expresiones

Stacks para manejo de ciclos y condicionales

`jumps_stack`: Administración de los saltos para las condicionales o ciclos, guarda el contador del cuádruplo a saltar o rellenar según sea el caso

`for_id_stack`: Almacena los ids de los ciclos para permitir anidación

Stack para manejo de arreglos

`dim_stack`: Almacena el id y la dimensión en la que se encuentra una variable dimensionada para permitir indexación por vectores.

4. Descripción de la Máquina Virtual

4.1 Equipo de computo, lenguaje y utilerías especiales usadas.

Como equipo de compu utilizamos nuestras computadoras personales (Laptops Mac)

Todo el proyecto se desarrolló en el lenguaje de programación **Python**.

Como editor de código fuente usamos Visual Studio Code

Dentro del proyecto, se utilizaron distintas herramientas para el desarrollo de la aplicación:

- Kivy: Librería para el desarrollo de aplicaciones móviles con Python

Además, se crearon utilerías nuevas para facilitar el desarrollo:

- Mapa de memoria de ejecución

4.2 Descripción detallada del proceso de administración de memoria en ejecución

4.2.1 Especificación gráfica de cada estructura de datos

Mapa de memoria

```
{
    dirección1: valor1,
    dirección2: valor2,
    ...
}
```

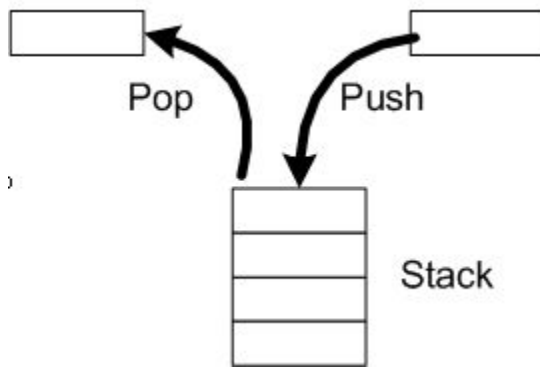
Se creó una clase para permitir varias instancias de la estructura conocida como mapa u objeto, con el objetivo de solamente almacenar en memoria lo que se utiliza. Esto se logra a través de una inicialización vacía del objeto, al cual se le van agregando como llaves las direcciones de memoria y como valor el valor almacenado en esa dirección, de forma que si alguna dirección no tiene valor esta no se crea en el objeto por lo cual no ocupa un espacio en la memoria.

Esta clase cuenta con las acciones de agregar un valor a la memoria (`set_value`) y obtener un valor de la memoria (`get_value`)

Al iniciar el programa, se crea una instancia que maneja la memoria global y otra que maneja la local.

La memoria local se agrega a un stack para permitir las llamadas a función y crear más instancias de memoria.

Manejo de stacks



Se utilizó la misma clase que en compilación

Stack de manejo de memoria

`memories_stack`: Almacena las memorias que están en espera de que termine la ejecución de una llamada a función, además de contener en su último elemento la memoria utilizada en ese momento.

Stack de manejo de llamada a función

`func_calls_stack`: Almacena el contador del último cuádruplo ejecutado para continuar la ejecución a partir de ahí cuando termine la llamada a una función.

4.2.2 Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución)

Los rangos de asignación para las direcciones de memoria fueron los siguientes:

- global
 - "int": 1000
 - "float": 4000
 - "char": 7000
- local
 - "int": 10000
 - "float": 13000
 - "char": 16000
- temporal
 - "int": 19000
 - "float": 21000
 - "char": 24000
- constante
 - "int": 27000
 - "float": 30000
 - "char": 33000
- pointer: 36000

No se hizo ninguna conversión para su almacenamiento en el mapa de memoria debido a que este se maneja como un mapa u objeto, donde la llave era la dirección de memoria que recibe del compilador.

5 Pruebas del funcionamiento del lenguaje

5.1 Pruebas de funcionamiento del proyecto

5.1.1 Codificación de la prueba

Sort

```
program Sort;
var
    int a[10];

main ( )
var int i, j, temp;
{
    for i = 1 to i < 5 do {
        a[i] = 5 - i;
        print('a', a[i]);
    }
    for i = 1 to i < 5 do {
        for j = i+1 to j < 5 do {
            if (a[j] < a[i]) {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    for i = 1 to i < 5 do {
        print('b', a[i]);
    }
}
```

Find

```
program Find;

func int find (int x)
var
    int a[10], i;
{
    for i = 0 to i < 10 do {
        a[i] = i;
    }

    for i = 0 to i < 10 do {
        if (a[i] == x){
            return (1);
        }
    }
    return (0);
}

main ( )
var int x;
{
```

```

        read(x);
        print(&find(x));
    }

```

Multiplicación de matrices

```
program Matrix;
```

```
func void mat ()
```

```
var
```

```
    int a[3][2], b[2][3], r[3][3], row, col, inner;
```

```
{
```

```
    a[0][0] = 1;
```

```
    a[0][1] = 4;
```

```
    a[1][0] = 2;
```

```
    a[1][1] = 5;
```

```
    a[2][0] = 3;
```

```
    a[2][1] = 6;
```

```
    b[0][0] = 7;
```

```
    b[0][1] = 8;
```

```
    b[0][2] = 9;
```

```
    b[1][0] = 10;
```

```
    b[1][1] = 11;
```

```
    b[1][2] = 12;
```

```
    for row = 0 to row < 3 do {
```

```
        for col = 0 to col < 3 do {
```

```
            r[row][col] = 0;
```

```
        }
```

```
    }
```

```
    for row = 0 to row < 3 do {
```

```
        for col = 0 to col < 3 do {
```

```
            for inner = 0 to inner < 2 do {
```

```
                r[row][col] = r[row][col] + (a[row][inner] *
```

```
b[inner][col]);
```

```
            }
```

```
            print(r[row][col]);
```

```
        }
```

```
    }
```

```
}
```

```
main ( )
```

```
{
```

```
    &mat();
```

```
}
```

Ciclo for

```
program FOR;
```

```
var
```

```
    int i, j, p, k;
```

```
    int Arreglo[10];
```

```
    float valor;
```

```
    int Matriz[10][8];
```

```

    char r;

func int for_loop (int j, char w)
var int i, r;
{
    r = 0;
    for i = 1 to i <= j do {
        r = r + i;
    }
    return (r);
}

main ( )
{
    print(&for_loop(5, 'a'));
}

```

Fibonacci iterativo

```

program Fibo_iter;
var
    int i, j, p, k;
    int Arreglo[10];
    float valor;
    int Matriz[10][8];
    char r;

func int fibbo (int j, char w)
var int i, aux, fib;
{
    aux = 1;
    fib = 0;
    for i = 1 to i <= j do {
        print(fib);
        aux = aux + fib;
        fib = aux - fib;
    }
    return (fib);
}

main ( )
var int r;
{
    read(r);
    &fibbo(r, 'a');
}

```

Fibonacci recursivo

```

program Fibo_rec;
var
    int i, j, p, k;

func int fibbo (int j )
{
    if (j <= 0) {
        return (0);
    }
}

```

```

        if (j == 1) {
            return (1);
        }

        return (&fibbo(j - 1) + &fibbo(j - 2));
    }

main ( )
var int r;
{
    read(r);
    print(&fibbo(r));
}

```

Factorial iterativo

program FACT_ITER;

```

main ( )
var
    int n, i;
    float factorial;
{
    factorial = 1.0;
    print("Enter a positive integer");
    read(n);

    if (n < 0) {
        print("Error! Factorial of a negative number doesn't
exist.");
    }
    else {
        for i = 1 to i <= n do {
            factorial = factorial * i;
        }
        print("Factorial of ", n, " = ", factorial);
    }
}

```

Factorial recursivo

program FACT_REC;

```

var
    int i, j, p, k;
    int Arreglo[10];
    float valor;
    int Matriz[10][8];
    char r;

func int fact (int j, char w)
var int i;
{
    if (j > 0) {
        return (j * &fact(j - 1, w));
    }
    else {
        return (1);
    }
}

```

```

    }
}

main ( )
{
    print(&fact(5, 'a'));
}

```

5.1.2 Resultados arrojados por la generación de código intermedio y por la ejecución.

Sort

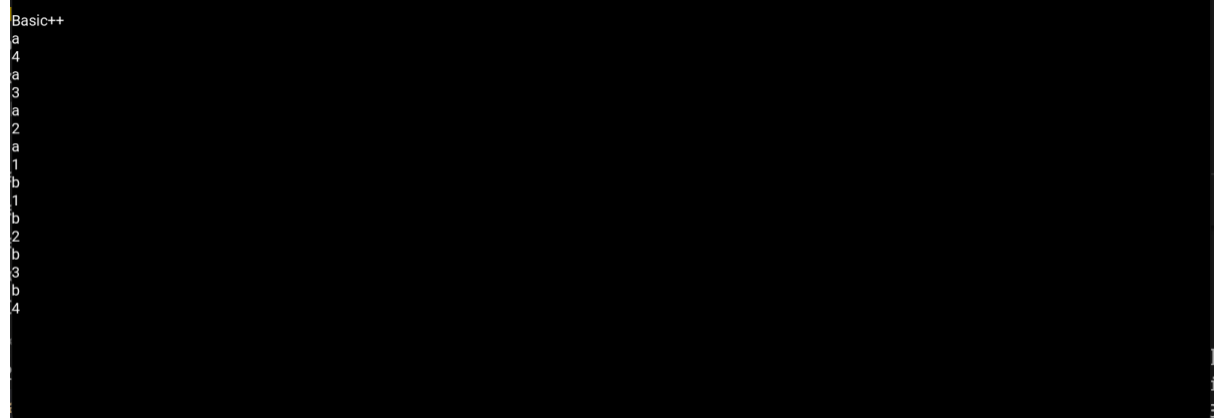
Output del código Intermedio:

```

[['goto', None, None, 1], ['=', 27000, None, 10000], ['<', 10000, 27002, 19000], ['gotoF', 19000,
None, 14], ['ver', 10000, None, 10], ['+', 10000, 27003, 36000], ['-', 27002, 10000, 19001], ['=',
19001, None, 36000], ['print', None, None, 33000], ['ver', 10000, None, 10], ['+', 10000,
27003, 36001], ['print', None, None, 36001], ['+', 10000, 27000, 10000], ['goto', None, None,
2], ['=', 27000, None, 10000], ['<', 10000, 27002, 19002], ['gotoF', 19002, None, 42], ['+',
10000, 27000, 19003], ['=', 19003, None, 10001], ['<', 10001, 27002, 19004], ['gotoF', 19004,
None, 40], ['ver', 10001, None, 10], ['+', 10001, 27003, 36002], ['ver', 10000, None, 10], ['+',
10000, 27003, 36003], ['<', 36002, 36003, 19005], ['gotoF', 19005, None, 38], ['ver', 10000,
None, 10], ['+', 10000, 27003, 36004], ['=', 36004, None, 10002], ['ver', 10000, None, 10], ['+',
10000, 27003, 36005], ['ver', 10001, None, 10], ['+', 10001, 27003, 36006], ['=', 36006, None,
36005], ['ver', 10001, None, 10], ['+', 10001, 27003, 36007], ['=', 10002, None, 36007], ['+',
10001, 27000, 10001], ['goto', None, None, 19], ['+', 10000, 27000, 10000], ['goto', None,
None, 15], ['=', 27000, None, 10000], ['<', 10000, 27002, 19006], ['gotoF', 19006, None, 51],
['print', None, None, 33001], ['ver', 10000, None, 10], ['+', 10000, 27003, 36008], ['print', None,
None, 36008], ['+', 10000, 27000, 10000], ['goto', None, None, 43], ['ENDFunc', None, None,
None]]

```

Output por la ejecución:



```

Basic++
a
4
a
3
a
2
a
1
b
1
b
2
b
3
b
4

```

Find

Output del código Intermedio:

```

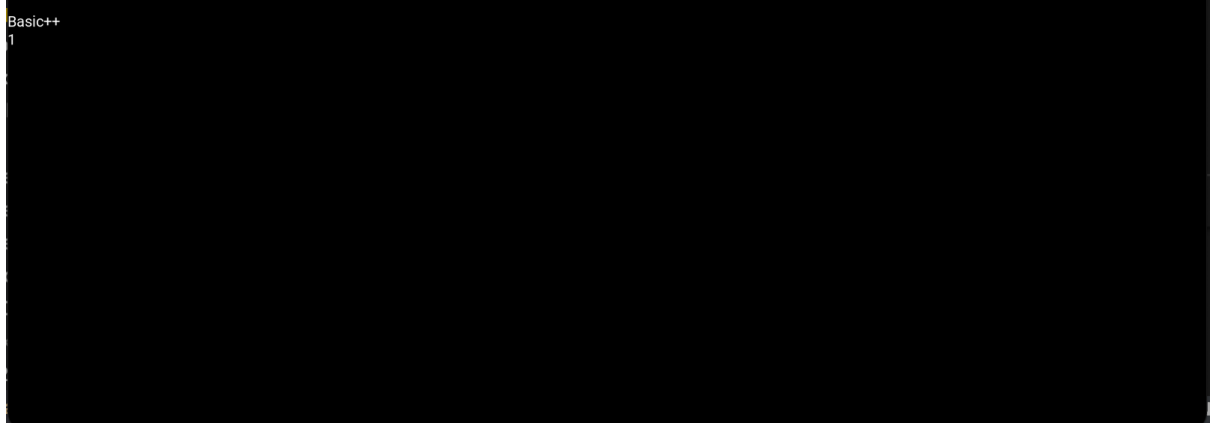
[['goto', None, None, 21], ['=', 27002, None, 10012], ['<', 10012, 27001, 19000], ['gotoF',
19000, None, 9], ['ver', 10012, None, 10], ['+', 10012, 27003, 36000], ['=', 10012, None,
36000], ['+', 10012, 27000, 10012], ['goto', None, None, 2], ['=', 27002, None, 10012], ['<',
10012, 27001, 19001], ['gotoF', 19001, None, 19], ['ver', 10012, None, 10], ['+', 10012, 27003,
36001], ['==', 36001, 10000, 19002], ['gotoF', 19002, None, 17], ['return', None, 'find', 27000],

```



```
['+', 10012, 27000, 10012], ['goto', None, None, 10], ['return', None, 'find', 27002], ['ENDFunc',
None, None, None], ['read', None, None, 10000], ['ERA', 'find', None, None], ['param', 10000,
0, 'find'], ['goSub', 'find', None, 1], ['=', 1000, None, 19000], ['print', None, None, 19000],
['ENDFunc', None, None, None]]
```

Output por la ejecución:



Multiplicación de matrices

Output del código Intermedio:

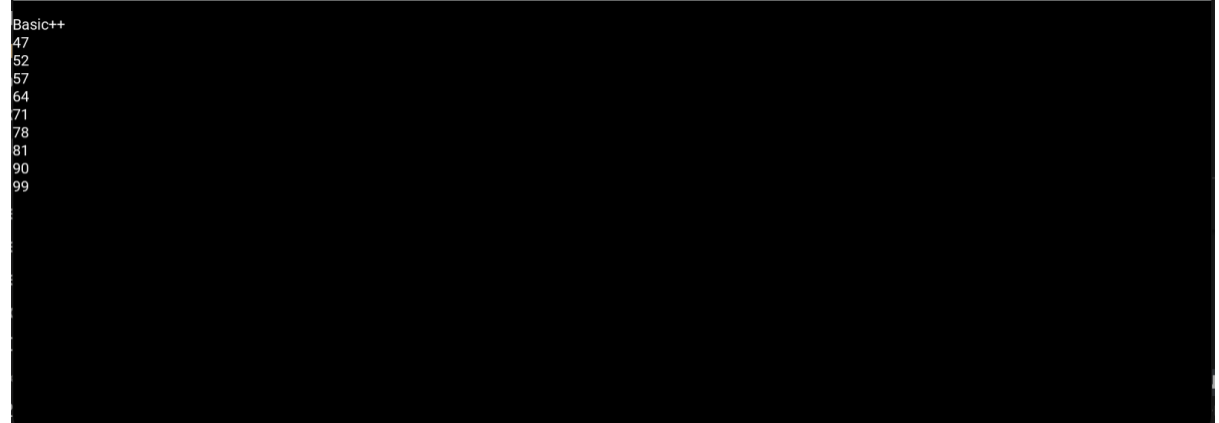
```
[['goto', None, None, 134], ['ver', 27003, None, 3], ['*', 27003,
27002, 19000], ['ver', 27003, None, 2], ['+', 19000, 27003, 19001],
['+', 19001, 27004, 36000], ['=', 27000, None, 36000], ['ver', 27003,
None, 3], ['*', 27003, 27002, 19002], ['ver', 27000, None, 2], ['+',
19002, 27000, 19003], ['+', 19003, 27004, 36001], ['=', 27005, None,
36001], ['ver', 27000, None, 3], ['*', 27000, 27002, 19004], ['ver',
27003, None, 2], ['+', 19004, 27003, 19005], ['+', 19005, 27004,
36002], ['=', 27002, None, 36002], ['ver', 27000, None, 3], ['*',
27000, 27002, 19006], ['ver', 27000, None, 2], ['+', 19006, 27000,
19007], ['+', 19007, 27004, 36003], ['=', 27006, None, 36003],
['ver', 27002, None, 3], ['*', 27002, 27002, 19008], ['ver', 27003,
None, 2], ['+', 19008, 27003, 19009], ['+', 19009, 27004, 36004],
['=', 27001, None, 36004], ['ver', 27002, None, 3], ['*', 27002,
27002, 19010], ['ver', 27000, None, 2], ['+', 19010, 27000, 19011],
['+', 19011, 27004, 36005], ['=', 27007, None, 36005], ['ver', 27003,
None, 2], ['*', 27003, 27001, 19012], ['ver', 27003, None, 3], ['+',
19012, 27003, 19013], ['+', 19013, 27008, 36006], ['=', 27009, None,
36006], ['ver', 27003, None, 2], ['*', 27003, 27001, 19014], ['ver',
27000, None, 3], ['+', 19014, 27000, 19015], ['+', 19015, 27008,
36007], ['=', 27010, None, 36007], ['ver', 27003, None, 2], ['*',
27003, 27001, 19016], ['ver', 27002, None, 3], ['+', 19016, 27002,
19017], ['+', 19017, 27008, 36008], ['=', 27011, None, 36008],
['ver', 27000, None, 2], ['*', 27000, 27001, 19018], ['ver', 27003,
None, 3], ['+', 19018, 27003, 19019], ['+', 19019, 27008, 36009],
['=', 27012, None, 36009], ['ver', 27000, None, 2], ['*', 27000,
27001, 19020], ['ver', 27000, None, 3], ['+', 19020, 27000, 19021],
['+', 19021, 27008, 36010], ['=', 27013, None, 36010], ['ver', 27000,
None, 2], ['*', 27000, 27001, 19022], ['ver', 27002, None, 3], ['+',
19022, 27002, 19023], ['+', 19023, 27008, 36011], ['=', 27014, None,
36011], ['=', 27003, None, 10021], ['<', 10021, 27001, 19024],
['gotoF', 19024, None, 89], ['=', 27003, None, 10022], ['<', 10022,
27001, 19025], ['gotoF', 19025, None, 87], ['ver', 10021, None, 3],
['*', 10021, 27001, 19026], ['ver', 10022, None, 3], ['+', 19026,
10022, 19027], ['+', 19027, 27015, 36012], ['=', 27003, None, 36012],
['+', 10022, 27000, 10022], ['goto', None, None, 77], ['+', 10021,
```

```

27000, 10021], ['goto', None, None, 74], ['=', 27003, None, 10021],
['<', 10021, 27001, 19028], ['gotoF', 19028, None, 133], ['=', 27003,
None, 10022], ['<', 10022, 27001, 19029], ['gotoF', 19029, None,
131], ['=', 27003, None, 10023], ['<', 10023, 27002, 19030],
['gotoF', 19030, None, 123], ['ver', 10021, None, 3], ['*', 10021,
27001, 19031], ['ver', 10022, None, 3], ['+', 19031, 10022, 19032],
['+', 19032, 27015, 36013], ['ver', 10021, None, 3], ['*', 10021,
27001, 19033], ['ver', 10022, None, 3], ['+', 19033, 10022, 19034],
['+', 19034, 27015, 36014], ['ver', 10021, None, 3], ['*', 10021,
27002, 19035], ['ver', 10023, None, 2], ['+', 19035, 10023, 19036],
['+', 19036, 27004, 36015], ['ver', 10023, None, 2], ['*', 10023,
27001, 19037], ['ver', 10022, None, 3], ['+', 19037, 10022, 19038],
['+', 19038, 27008, 36016], ['*', 36015, 36016, 19039], ['+', 36014,
19039, 19040], ['=', 19040, None, 36013], ['+', 10023, 27000, 10023],
['goto', None, None, 96], ['ver', 10021, None, 3], ['*', 10021,
27001, 19041], ['ver', 10022, None, 3], ['+', 19041, 10022, 19042],
['+', 19042, 27015, 36017], ['print', None, None, 36017], ['+',
10022, 27000, 10022], ['goto', None, None, 93], ['+', 10021, 27000,
10021], ['goto', None, None, 90], ['ENDFunc', None, None, None],
['ERA', 'mat', None, None], ['goSub', 'mat', None, 1], ['ENDFunc',
None, None, None]]

```

Output por la ejecución:



Ciclo for

Output del código Intermedio:

```

[['goto', None, None, 11], ['=', 27003, None, 10002], ['=', 27000,
None, 10001], ['<=', 10001, 10000, 19000], ['gotoF', 19000, None, 9],
['+', 10002, 10001, 19001], ['=', 19001, None, 10002], ['+', 10001,
27000, 10001], ['goto', None, None, 3], ['return', None, 'for_loop',
10002], ['ENDFunc', None, None, None], ['ERA', 'for_loop', None,
None], ['param', 27004, 0, 'for_loop'], ['param', 33000, 1,
'for_loop'], ['goSub', 'for_loop', None, 1], ['=', 1095, None,
19000], ['print', None, None, 19000], ['ENDFunc', None, None, None]]

```

Output por la ejecución:

```
Basic++
15
```

Fibonacci iterativo

Output del código Intermedio:

```
[['goto', None, None, 11], ['=', 27003, None, 10002], ['=', 27000, None, 10001], ['<=', 10001, 10000, 19000], ['gotoF', 19000, None, 9], ['+', 10002, 10001, 19001], ['=', 19001, None, 10002], ['+', 10001, 27000, 10001], ['goto', None, None, 3], ['return', None, 'fibbo', 10002], ['ENDFunc', None, None, None], ['ERA', 'fibbo', None, None], ['param', 27004, 0, 'fibbo'], ['param', 33000, 1, 'fibbo'], ['goSub', 'fibbo', None, 1], ['=', 1095, None, 19000], ['print', None, None, 19000], ['ENDFunc', None, None, None]]
```

Output por la ejecución:

```
Basic++
0
1
1
2
3
```

Fibonacci recursivo

Output del código Intermedio:

```
[['goto', None, None, 20], ['<=', 10000, 27001, 19000], ['gotoF', 19000, None, 4], ['return', None, 'fibbo', 27001], ['==', 10000, 27000, 19001], ['gotoF', 19001, None, 7], ['return', None, 'fibbo', 27000], ['ERA', 'fibbo', None, None], ['- ', 10000, 27000, 19002], ['param', 19002, 0, 'fibbo'], ['goSub', 'fibbo', None, 1], ['=', 1004, None, 19003], ['ERA', 'fibbo', None, None], ['- ', 10000, 27002, 19004], ['param', 19004, 0, 'fibbo'], ['goSub', 'fibbo', None, 1], ['=', 1004, None, 19005], ['+', 19003, 19005, 19006], ['return', None, 'fibbo', 19006], ['ENDFunc', None, None, None], ['read', None, None, 10000], ['ERA', 'fibbo', None, None], ['param', 10000, 0, 'fibbo'], ['goSub', 'fibbo', None, 1], ['=', 1004, None, 19000], ['print', None, None, 19000], ['ENDFunc', None, None, None]]
```

Output por la ejecución:

```
Basic++
3
```

Factorial iterativo

Output del código Intermedio:

```
['goto', None, None, 1], ['=', 27000, None, 13000], ['print', None,
None, 33000], ['read', None, None, 10000], ['<', 10000, 27001,
19000], ['gotoF', 19000, None, 8], ['print', None, None, 33001],
['goto', None, None, 19], ['=', 27000, None, 10001], ['<=', 10001,
10000, 19001], ['gotoF', 19001, None, 15], ['*', 13000, 10001,
21000], ['=', 21000, None, 13000], ['+', 10001, 27000, 10001],
['goto', None, None, 9], ['print', None, None, 33002], ['print',
None, None, 10000], ['print', None, None, 33003], ['print', None,
None, 13000], ['ENDFunc', None, None, None]]
```

Output por la ejecución:

```
Basic++
Enter a positive integer
Factorial of
5
=
120.0
```

Factorial recursivo

Output del código Intermedio:

```
[['goto', None, None, 14], ['>', 10000, 27003, 19000], ['gotoF',
19000, None, 12], ['ERA', 'fact', None, None], ['-', 10000, 27000,
19001], ['param', 19001, 0, 'fact'], ['param', 16000, 1, 'fact'],
['goSub', 'fact', None, 1], ['=', 1095, None, 19002], ['*', 10000,
19002, 19003], ['return', None, 'fact', 19003], ['goto', None, None,
13], ['return', None, 'fact', 27000], ['ENDFunc', None, None, None],
['ERA', 'fact', None, None], ['param', 27004, 0, 'fact'], ['param',
33000, 1, 'fact'], ['goSub', 'fact', None, 1], ['=', 1095, None,
19000], ['print', None, None, 19000], ['ENDFunc', None, None, None]]
```

Output por la ejecución:

Basic++
120

6. Documentación del código del proyecto

```
# Add th variable to the variable table
def p_add_id(p):
    '''add_id : '''
    global current_id, current_type, current_func, global_var_table,
    local_var_table, address, counter
    current_id = p[-1]
    if context == 'global':
        if current_id not in global_var_table:
            global_var_table[current_id] = [
                current_id, current_type,
                address['global'][current_type] + counter['global'][current_type],
                []
            ]
            counter['global'][current_type] += 1
        else:
            print('ERROR: Variable already defined', current_id)
            raise CompilerError(f'ERROR: Variable already defined
{current_id}')

    else:
        if current_id not in local_var_table:
            local_var_table[current_id] = [
                current_id, current_type,
                address['local'][current_type] + counter['local'][current_type], []
            ]
            counter['local'][current_type] += 1
        else:
            print('ERROR: Variable already defined', current_id)
            raise CompilerError(f'ERROR: Variable already defined
{current_id}')

# Add the verification quadruple and the operations to get the value
in memory according to the dimension for the first dimension
def p_verify_quad_1(p):
    '''verify_quad_1 : '''
    global global_var_table, local_var_table, elements_stack,
    quadruples, current_arr_id, dim_stack, constant_var_table
    dims = []
    first_dim = 0
    current_arr_id = dim_stack.peek()['id']
    if current_arr_id in local_var_table:
        dims = local_var_table[current_arr_id][3]
        first_dim = dims[0]
        quadruples.append(['ver', elements_stack.peek(), None,
first_dim])
    elif current_arr_id in global_var_table:
        dims = global_var_table[current_arr_id][3]
        first_dim = dims[0]
        quadruples.append(['ver', elements_stack.peek(), None,
first_dim])

    if len(dims) > 1:
        element_op = elements_stack.pop()
        element_type = types_stack.pop()
        result_type = semantic_cube[element_type]['*']['int']
```

```

        if result_type != None:
            result = address['temp'][result_type] + \
                counter['temp'][result_type]
            counter['temp'][result_type] += 1

            quadruples.append(['*', element_op,
constant_var_table[dims[1]][2], result])

            elements_stack.push(result)
            types_stack.push(result_type)
        else:
            print("ERROR: Type mismatch", element_op, '*', dims[1])
            raise CompilerError(f"ERROR: Type mismatch {element_op},
 '*', {dims[1]}")

# Add the verification quadruple and the operations to get the value
in memory according to the dimension for the second dimension
def p_verify_quad_2(p):
    '''verify_quad_2 : '''
    global global_var_table, local_var_table, elements_stack,
quadruples, current_arr_id, dim_stack, constant_var_table
    dims = []
    second_dim = 0
    current_arr_id = dim_stack.peek()['id']
    if current_arr_id in local_var_table:
        dims = local_var_table[current_arr_id][3]
        second_dim = dims[1]
        quadruples.append(['ver', elements_stack.peek(), None,
second_dim])
    elif current_arr_id in global_var_table:
        dims = global_var_table[current_arr_id][3]
        second_dim = dims[1]
        quadruples.append(['ver', elements_stack.peek(), None,
second_dim])

    aux2 = elements_stack.pop()
    type_aux2 = types_stack.pop()
    aux1 = elements_stack.pop()
    type_aux1 = types_stack.pop()
    result_type = semantic_cube[type_aux2]['+'][type_aux1]

    if result_type != None:
        result = address['temp'][result_type] +
counter['temp'][result_type]
        counter['temp'][result_type] += 1

        quadruples.append(['+', aux1, aux2, result])

        elements_stack.push(result)
        types_stack.push(result_type)
    else:
        print("ERROR: Type mismatch", aux1, '+', aux2)
        raise CompilerError(f"ERROR: Type mismatch, {aux1}, '+',
{aux2}")

```

```

# Add base address to the pointer variable in memory
def p_add_base(p):
    '''add_base : '''
    global elements_stack, types_stack, address, counter,
    current_arr_id, global_var_table, local_var_table, current_id
    element_op = elements_stack.pop()
    element_type = types_stack.pop()
    result_type = semantic_cube[element_type]['+']['int']

    base_address = 0
    if result_type != None:
        result = address['pointer'] + counter['pointer']
        counter['pointer'] += 1

        if current_arr_id in local_var_table:
            base_address = local_var_table[current_arr_id][2]
        elif current_arr_id in global_var_table:
            base_address = global_var_table[current_arr_id][2]

        if base_address not in constant_var_table:
            constant_var_table[base_address] = (
                base_address, 'int', address['constant']['int'] +
counter['constant']['int'])
            counter['constant']['int'] += 1

        quadruples.append(['+', element_op,
constant_var_table[base_address][2], result])

        elements_stack.push(result)
        types_stack.push(result_type)
        current_id = current_arr_id
    else:
        print("ERROR: Type mismatch", element_op, '+', base_address)
        raise CompilerError(f"ERROR: Type mismatch, {element_op},
'+', {base_address}")

    if not dim_stack.is_empty():
        dim_stack.pop()

# Call function inside expression
def p_call_func_exp(p):
    '''call_func_exp : AMP ID call_func_era L_P args R_P'''
    global current_call, dir_func, k
    if len(dir_func[current_call]['param_types']) == 0 or k ==
(len(dir_func[current_call]['param_types'])-1):
        quadruples.append(['goSub', current_call, None,
            dir_func[current_call]['start_quad']])
    if current_call in global_var_table:
        func_var = global_var_table[current_call]
        func_temp_add = address['temp'][func_var[1]] + \
            counter['temp'][func_var[1]]
        counter['temp'][func_var[1]] += 1

```



```

        quadruples.append(['=', func_var[2], None,
func_temp_add])

        elements_stack.push(func_temp_add)
        types_stack.push(func_var[1])
    else:
        print('ERROR: Cannot call void function on expresion',
current_call)
        raise CompilerError(f'ERROR: Cannot call void function
on expresion {current_call}')
    else:
        r = len(dir_func[current_call]['param_types'])-1
        print('ERROR: Missing arguments', k, len(
            dir_func[current_call]['param_types'])-1)
        raise CompilerError(f'ERROR: Missing arguments {k}, {r}')

# Add one to the id for the next iteration
def p_for_statement(p):
    '''for_statement : FOR id for_id EQUAL expression for_id_quad TO
breadcrumb expression exp_type do_statement'''
    global jumps_stack, quadruples, local_var_table,
global_var_table, constant_var_table, current_for_id
    end = jumps_stack.pop()
    element = None
    return_jump = jumps_stack.pop()
    current_for_id = for_id_stack.pop()
    if current_for_id in local_var_table:
        element = local_var_table[current_for_id][2]
        id_type = local_var_table[current_for_id][1]
    elif current_for_id in global_var_table:
        element = global_var_table[current_for_id][2]
        id_type = global_var_table[current_for_id][1]

    if element != None:
        result_type = semantic_cube[id_type]['+']['int']
        if result_type != None:
            quadruples.append(
                ['+', element, constant_var_table[1][2], element])
        else:
            print("ERROR: Type mismatch")
            raise CompilerError(f"ERROR: Type mismatch, {element},
'+', {constant_var_table[1][2]}")
    else:
        print('ERROR: Undeclared variable', current_for_id)
        raise CompilerError(f'ERROR: Undeclared variable
{current_for_id}')
    quadruples.append(['goto', None, None, return_jump])
    fill(end, len(quadruples))

# Run the program, iterating through the quadruples list doing the
corresponding instructions
def run(instruction_pointer=0):
    current_quad = quadruples[instruction_pointer]
    instruction = ''
    first_element = 0

```

```

second_element = 0
third_element = 0
memories_stack.push(new_memory)

def igoto():
    global instruction_pointer
    instruction_pointer = third_element
    return instruction_pointer

def igotoF():
    global instruction_pointer
    first_value = get_value(first_element)
    instruction_pointer = third_element if first_value == 0 else
instruction_pointer + 1
    return instruction_pointer

def iERA():
    global instruction_pointer, new_memory, in_ERA
    new_memory = Memory()
    in_ERA = True
    instruction_pointer += 1
    return instruction_pointer

def iparam():
    global instruction_pointer, counter, base_address,
new_memory
    value = get_value(first_element)
    param_type =
dir_func[third_element]['param_types'][second_element]
    address = base_address[param_type] + counter[param_type]
    counter[param_type] += 1
    new_memory.set_value(value, address)
    instruction_pointer += 1
    return instruction_pointer

def igoSub():
    global instruction_pointer, counter, in_ERA, memories_stack,
new_memory
    counter = {
        "int": 0,
        "float": 0,
        "char": 0
    }
    instruction_pointer += 1
    func_calls_stack.push(instruction_pointer)
    in_ERA = False
    memories_stack.push(new_memory)
    instruction_pointer = third_element
    return instruction_pointer

def iEndFunc():
    global instruction_pointer, memories_stack
    if not memories_stack.is_empty():
        memories_stack.pop()
    if not func_calls_stack.is_empty():

```

```

        instruction_pointer = func_calls_stack.pop()
    else:
        instruction_pointer += 1
    return instruction_pointer

def iprint():
    global instruction_pointer

program.display_output(get_value(third_element), display_name="VM")
    instruction_pointer += 1
    return instruction_pointer

def iread():
    global instruction_pointer
    if value := program.get_stdoutin():
        program.text_input_box.size_hint_y = None
        program.text_input_box.height = '0dp'
        value_type = get_type(value)
        if check_range(third_element, value_type):
            set_value(value, third_element)
            instruction_pointer += 1
        else:
            print('ERROR: Invalid input type', value,
third_element)
            program.display_output(f'ERROR: Invalid input type:
{value}', display_name="VM")
    return instruction_pointer

def ireturn():
    global instruction_pointer, memories_stack
    value = get_value(third_element)
    current_func = second_element
    memories_stack.pop()
    set_value(value, dir_func[current_func]['memory_address'])
    instruction_pointer = func_calls_stack.pop()
    return instruction_pointer

def iver():
    global instruction_pointer
    value = get_value(first_element)
    if value in range(third_element):
        instruction_pointer += 1
    else:
        print('ERROR: Index out of bounds', value)
        program.display_output(f'ERROR: Index out of bounds:
{value}', display_name="VM")
    return instruction_pointer

def iassign():
    global instruction_pointer
    first_value = get_value(first_element)
    set_value(first_value, third_element if third_element <
36000 else get_memory_value(third_element))
    instruction_pointer += 1
    return instruction_pointer

```

```

def iexp():
    global instruction_pointer
    first_value = get_value(first_element)
    second_value = get_value(second_element)
    result = get_result(first_value, instruction, second_value)

    set_value(result, third_element)
    instruction_pointer += 1
    return instruction_pointer

def instruction_error():
    print('ERROR: Wrong instruction')
    program.display_output(f'ERROR: Wrong instruction',
display_name="VM")

instruction_switch = {
    'goto': igoto,
    'gotoF': igotoF,
    'goSub': igoSub,
    'param': iparam,
    'ERA': iERA,
    'ENDFunc': iEndFunc,
    'print': iprint,
    'read': iread,
    'return': ireturn,
    'ver': iver,
    '=': iassign,
    '+': iexp,
    '-': iexp,
    '*': iexp,
    '/': iexp,
    '%': iexp,
    '<': iexp,
    '>': iexp,
    '<=': iexp,
    '>=': iexp,
    '!=': iexp,
    '==': iexp,
    '&&': iexp,
    '||': iexp,
}

while instruction_pointer < len(quadruples):
    current_quad = quadruples[instruction_pointer]
    instruction = current_quad[0]
    first_element = current_quad[1]
    second_element = current_quad[2]
    third_element = current_quad[3]
    previous_instruction_pointer = instruction_pointer

    instruction_pointer = instruction_switch.get(instruction,
instruction_error) ()

    if previous_instruction_pointer == instruction_pointer:

```

```
program.save_state(instruction_pointer)
break
```