Diseño de Compiladores

Grupo 1

Profesores:

Elda Guadalupe Quiroga González

Héctor Gibrán Ceballos Cancino

TapiCompi

A01234029 Jazmín Yolistli Santibáñez de la Rosa

22 de noviembre de 2022

Índice

1 Acerca del proyecto

1.1 Propósito y alcance del proyecto

El propósito de este proyecto es hacer un compilador y máquina virtual para un lenguaje capaz de hacer análisis estadísticos significativos, es decir, que pueda hacer modelos estadísticos básicos y sus gráficas, similar al lenguaje R.

1.2 Análisis de requerimientos

El proyecto debe cumplir con las siguientes características:

- Está formado por 2 partes fundamentales: el compilador y la máquina virtual.
- El compilador debe ser capaz de realizar análisis léxico, sintáctico y semántico.
- Si ocurre un error dentro de los análisis mencionados, se debe mostrar un mensaje de error descriptivo que permita entender por qué ocurrió el fallo.
- El lenguaje creado debe contar con los siguientes elementos:
  - Expresiones matemáticas: Aritméticas, lógicas y relacionales.
  - Estatutos lineales: Asignación, lectura, escritura
  - Estatutos condicionales: Ciclos, decisiones
  - Módulos: Funciones con y sin retorno, con parámetros.
  - Variables locales y globales.
  - Elementos estructurados: Vectores y matrices.
- Como producto final de la compilación, se debe producir código intermedio en formato de cuádruplos. Los cuádruplos deben hacer referencia a direcciones virtuales que representan el almacenamiento en la memoria de los símbolos leídos o generados.
- La máquina virtual debe poder recibir los cuádruplos y ejecutarlos para reproducir el código escrito por el usuario.
- La máquina virtual debe detectar errores de ejecución como divisiones sobre cero, límites de arreglos, entre otros.

1.2.1 Principales casos de prueba

Se realizarán casos para probar principalmente los elementos fundamentales del lenguaje, revisando que las expresiones matemáticas generen resultados correctos, que los estatutos lineales cumplan su propósito al ser llamadas, que los estatutos

condicionales tengan un correcto control de flujo de acuerdo a la expresión evaluada, que los arreglos accedan al valor correspondiente real, que las funciones tengan su memoria local y que el cambio entre contextos se realice correctamente.

Revisar la sección 5. Pruebas de funcionamiento donde se describen ejemplos de código.

1.3 Descripción del proceso de desarrollo

Para el desarrollo del proyecto se utilizó github como repositorio y control de versiones. Puede ser consultado en JazminSantibanez/TapiCompi.

Para la organización y toma de notas para el diseño del compilador se creó una página de Notion, la cual se puede visitar a través del siguiente enlace Tablero TapiCompi.

1.3.1 Bitácoras semanales

Avance 0

Fecha de entrega: Sábado 8 de Octubre de 2022
Se desarrolló la propuesta final del proyecto la cual es la definición del lenguaje. Se definieron los tokens (tanto palabras reservadas como definiciones simples de tokens), los diagramas de sintaxis junto con su gramática formal y las principales consideraciones semánticas.

Avance 1

Fecha de entrega: Lunes 3 de Octubre de 2022
Inicié la codificación del léxico y sintaxis en PLY para Python. Terminé el léxico pero sólo hice el equivalente de 2 diagramas de sintaxis porque no tuve mucho tiempo durante la semana.

Avance 2

Fecha de entrega: Miércoles 12 de Octubre de 2022
Terminé de escribir la gramática formal en código y realicé algunos ajustes en los tokens. Además, implementé el cubo semántico (usando un diccionario de diccionarios y una función para poder revisar el emparejamiento de los tipos).

Avance 3

Fecha de entrega: Lunes 17 de Octubre de 2022
Se corrigieron todos los errores de recursión y *shift/reduce* de la gramática. Se pueden leer archivos de prueba y hacer el análisis sintáctico sin warnings.
Se comenzó a hacer un boceto del directorio de funciones y tablas de variables pero no se pasó a código por el momento.

Avance 4

Fecha de entrega: Viernes 4 de Noviembre de 2022
Esta semana trabajé en el desarrollo del directorio de funciones y tablas de variables. La declaración de variables y funciones funciona correctamente (detecta doble declaración).
Se generan cuádruplos correctos para los estatutos secuenciales: asignación, leer y escribir.
Se generan cuadruplos correctos para expresiones aritméticas básicas (sumar, restar, multiplicar y dividir), incluyendo paréntesis.

Avance 5

Fecha de entrega: Lunes 7 de Noviembre de 2022
Se añadieron los puntos neurálgicos para decisiones (if) y ciclos (while). Se producen cuádruplos correctos para los estatutos mencionados.
Modifiqué el formato de impresión de los cuádruplos para que se mostrarán de forma más organizada.
Terminé los puntos neurálgicos y generación de cuádruplos para los operadores lógicos y relacionales.

Avance 6

Fecha de entrega: Martes 15 de Noviembre de 2022
Se agregó do-while al léxico y semántica. Se generan cuádruplos para este estatuto.
Se generan cuadruplos correctos para las funciones, así como la dirección de las funciones.
Comencé la implementación del segmento de memoria (utilizando una clase que genera arreglos para cada tipo local y temporal, recibiendo como parámetro la cantidad de cada recurso) y del manejo de direcciones virtuales.

Avance 7

Fecha de entrega: Lunes 21 de Noviembre de 2022

Se creó la tabla de constantes (diccionario optimizado a búsqueda valor:dirección), se integró al análisis semántico para detectar constantes y guardarlas. Se añadió el uso de direcciones virtuales de las constantes para la máquina virtual. Implementación de máquina virtual para los códigos de operación de expresiones aritméticas (y lógicas y relaciones), de estatutos secuenciales, ciclos, decisiones y módulos (con recursión). Se desarrolló la memoria y el contexto en la máquina virtual para poder hacer cambio de memorias entre llamadas de funciones.

1.3.2 Lista de commits

La lista de cambios subidos al repositorio (hasta el momento de la generación de este documento) se encuentra al final del documento en el anexo 1.

1.3.3 Reflexión

El aprendizaje más grande que me llevo de este proyecto es el de ir manejando mis tiempos y hacer avances consistentes cuando esté haciendo proyectos de gran tamaño. Es fácil dejarse llevar y pensar que se puede hacer rápido pero con trabajo constante salen las cosas con mejor calidad y sin prisas. En cuanto a aprendizajes técnicos, ahora entiendo más como trabajar con Python, en especial la parte de funciones, acceso a diccionarios, creación de clases e importar archivos completos, variables o funciones. En conclusión, este proyecto fue un mini repaso exprés de todo lo que vi en la carrera pues tenía que entender bien qué estaba pasando para poder implementar las funcionalidades de lo que quería hacer, pero ahora entiendo mejor cómo funciona un compilador y todo el proceso que ocurre cuando ejecutamos nuestros programas.

2 Acerca del lenguaje

2.1 Nombre del Lenguaje

TapiCompi

2.2 Principales características

TapiCompi se basa en el funcionamiento base de lenguajes como c. El lenguaje permite:

- Crear variables y constantes de tipo int, float, bool y char.
- Manejar entrada y salida de datos (instrucciones de *input* a variables e impresión a consola).
- Ejecutar operaciones aritméticas, lógicas, relacionales.
- Usar estatutos de decisión (if, if-else) y ciclos (while, do-while).
- Declarar funciones con o sin retorno y hacer uso de llamadas a las mismas.
- Declarar arreglos en una y dos dimensiones.
- Hacer funciones básicas de estadística: suma, resta, mínimo, máximo, promedio, desviación estándar, varianza y rango.

2.3 Lista de errores

A continuación se presentan los errores definidos que pueden ocurrir en las etapas de compilación y ejecución:

2.3.1 Errores en compilación

Hace referencia a los errores que son detectados durante el análisis léxico, sintáctico y/o semántico:

- Usar tokens no reconocidos.
- No seguir la estructura de construcción para cada estatuto (error de sintaxis)
- Declaración múltiple de variables: Intentar declarar una variable que ya fue declarada anteriormente (Pueden existir una variable en global y otra con el mismo nombre dentro de una función, siendo la excepción que se declare en la función de main).

- Declaración múltiple de funciones: Sólo puede existir una función a la vez con el mismo nombre.
- Una variable global y función con retorno no pueden tener el mismo nombre.
- Hacer uso de una variable que no ha sido declarada.
- Realizar una operación aritmética con tipos incompatibles (Ejemplo: suma con booleanos)
- Discrepancia en el número de paréntesis usados.
- El resultado de una evaluación para condicionales no es booleano.
- Llamada a una función que no ha sido declarada.
- Llamada a una función con más argumentos de los que fue declarada.
- Llamada a una función con menos argumentos de los que fue declarada.
- Tipos incompatibles en los argumentos dentro de la llamada de una función y los parámetros con los que fue declarada.
- Estatuto de regreso (return) dentro de una función void.
- Discrepancia en el tipo de retorno de una función y el resultado de la expresión dentro del estatuto *return*

2.3.2 Errores en ejecución

Hace referencia a los errores que son detectados por la máquina virtual al ejecutar las instrucciones de los cuádruplos.

- Divisiones entre cero.
- Demasiadas llamadas a funciones encadenadas (el stack de contextos es mayor o igual a 200).
- El tamaño del resultado de una expresión excede el tamaño de una casilla de la memoria. (Memory overflow)
- Se intenta acceder a una casilla de un arreglo fuera de los límites con los que fue declarado.
- Se llama a un arreglo con más dimensiones (o menos) de las que fue declarado.

3 Acerca del compilador

3.1 Equipo de computo, lenguaje y librerías usadas

El compilador fue realizado en un equipo con sistema operativo Windows. Se desarrolló en Python con las siguientes librerías:

- PLY: Para el parser y lexer.
- deque de la librería collections: Para la implementación de pilas.
- pandas: Para pasar los cuádruplos a formato de *dataframe*.
- numpy: Para hacer uso de enteros de 64 bits.

3.2 Léxico

3.2.1 Tokens

Palabras reservadas:

```
## <PROGRAMA>
‘program’ : ‘PROGRAM’
'main' : 'MAIN',

## <DEC_VAR>
'var' : 'VAR',

## <TIPO_S>
'int' : 'INT',
'float' : 'FLOAT',
'char' : 'CHAR',
‘bool’ : ‘BOOL’,

## <TIPO_C>
‘dataframe’ : ‘DATAFRAME’
'file' : 'FILE'

## <FUNCS>
'func' : 'FUNC',
'void' : 'VOID',
'return' : 'RETURN',

## <ESTATUTOS>
'read' : 'READ',
```

'print' : 'PRINT',

'if' : 'IF',
'else' : 'ELSE',

'while' : 'WHILE',
‘do’ : ‘DO’,

Definiciones simples de tokens:

OP_ASSIGN : ‘=’

Logical operators
OP_AND = ‘\&’
OP_OR = ‘\|’

Airthmetic operators
OP_ADD: ‘+’
OP_SUBTR : ‘-’
OP_MULT : ‘*’
OP_DIV : ‘/’

Relational operators
OP_EQ : ‘==’
OP_NEQ : ‘!=’
OP_LT : ‘<’
OP_GT : ‘>’
OP_LTE : ‘<=’
OP_GTE : ‘>=’

Separators
lPAREN : ‘(’
rPAREN : ‘)’
lBRACE : ‘{‘
rBRACE : ‘}’
lBRACKET : ‘[‘
rBRACKET : ‘]’

SEP_SEMICOLON : ‘;’
SEP_COMMA : ‘,’
SEP_COLON : ‘:’
ignore : ‘\t’

3.2.2 Expresiones regulares

Expresiones regulares para tokens compuestos:

COMENTARIO : '\#.*'
ID : '[a-zA-Z][a-zA-Z0-9_]*'
CTE_F : '\d+\.\d+'
CTE_I : '[0-9]+'
CTE_CHAR : ' (\'[a-zA-Z0-9]\') '
LETRERO : ' r'"[^"]*" '

3.3 Sintaxis: Gramática formal

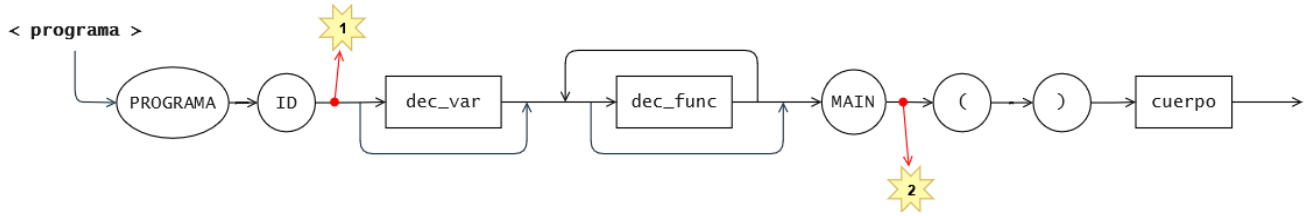
```
programa : PROGRAM ID SEP_COLON aux_prog aux_prog2 MAIN lPAREN
rPAREN cuerpo;
aux_prog : dec_var
         | ε;
aux_prog2 : dec_func aux_prog2
          | ε;

cuerpo : lBRACE aux_cuerpo bloque rBRACE;
aux_cuerpo : dec_var
           | ε;

bloque : estatuto bloque
       | ε;

dec_var : VAR aux_dv;
aux_dv : aux_dv2 aux_dv3
       | aux_dv2 aux_dv3 aux_dv;
aux_dv2 : tipo_s;
aux_dv3 : ID aux_dv4 aux_dv6;
aux_dv4 : arr aux_dv5
        | ε;
aux_dv5 : arr
        | ε;
aux_dv6 : SEP_COMMA aux_dv3
        | ε;

tipo_s : INT
       | FLOAT
       | CHAR
```

```
       | BOOL;
arr : lBRACKET aux_arr rBRACKET;
     | h_exp;

call_var : ID aux_cv;
aux_cv : arr aux_cv2
        | ε;
aux_cv2 : arr
        | ε;

dec_func : FUNC aux_df ID lPAREN aux_df2 rPAREN cuerpo;
aux_df : VOID
       | tipo_s;
aux_df2 : params
        | ε;

params : tipo_s ID aux_cv aux_params;
aux_params : SEP_COMMA params
           | ε;

call_func : ID lPAREN aux_cf rPAREN
aux_cf : h_exp aux_cf2
       | ε;
aux_cf2 : SEP_COMMA aux_cf
        | ε;

estatuto : asignacion
         | call_func
         | leer
         | escribir
         | return
         | condicion
         | ciclo_while
         | ciclo_do_while
         | COMENTARIO;

return : RETURN lPAREN h_exp rPAREN;

asignacion : call_var OP_ASSIGN h_exp;

leer : READ lPAREN aux_leer rPAREN;
aux_leer : call_var
         | call_var SEP_COMMA aux_leer;
```

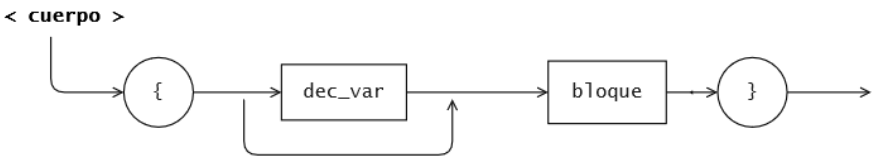
```
escribir : PRINT lPAREN aux_escribir rPAREN
aux_escribir : aux_escribir2
             | aux_escribir2 SEP_COMMA aux_escribir
aux_escribir2 : h_exp
              | LETRERO
              | CTE_CHAR

condicion : IF lPAREN h_exp rPAREN lBRACE bloque rBRACE aux_cond;
aux_cond : ELSE lBRACE bloque rBRACE
         | ε;

ciclo_while : WHILE lPAREN h_exp rPAREN lBRACE bloque rBRACE;

ciclo_do_while : DO lBRACE bloque rBRACE WHILE lPAREN h_exp rPAREN;

h_exp : s_exp
      | s_exp aux_h_exp h_exp
aux_h_exp : OP_AND
          | OP_OR

s_exp : exp
      | exp aux_s_exp s_exp
aux_s_exp : OP_EQ
          | OP_NEQ
          | OP_GT
          | OP_LT
          | OP_GTE
          | OP_LTE

exp : termino
    | termino aux_exp exp
aux_exp : OP_ADD
        | OP_SUBTR

termino : factor
        | factor OP_MULT termino
        | factor OP_DIV termino

factor : lPAREN h_exp rPAREN
       | CTE_I
       | CTE_F
       | call_var
       | call_func
```

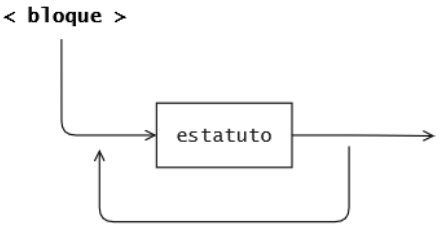
3.4 Semántica y código intermedio

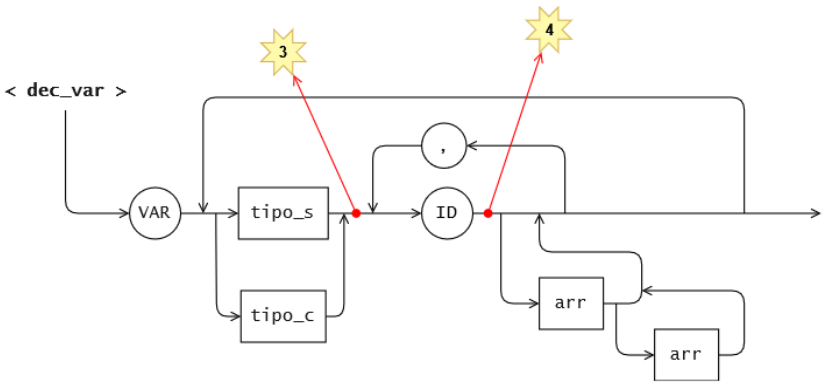
3.4.1 Diagramas de sintaxis



1. Se crea el directorio de funciones y se añade la función global con tipo void. Crear la tabla de variables globales ligada a la función global y guardar el nombre del programa en variables globales. Scope = ‘global’
2. Hacer pop del stack de saltos, ir al cuádruplo y poner el número del quad_pointer. Scope = ‘global’ y function_type = ‘void’

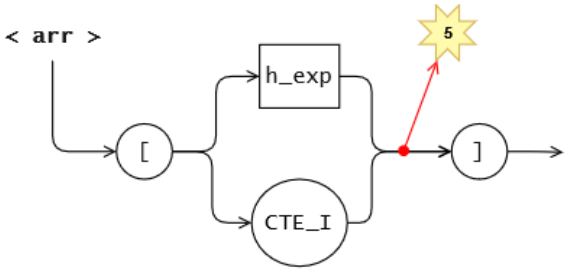




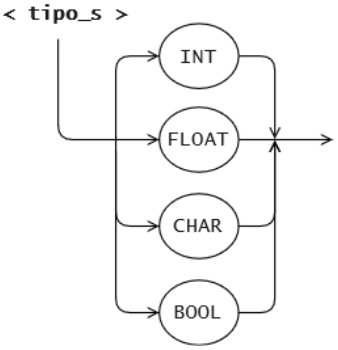


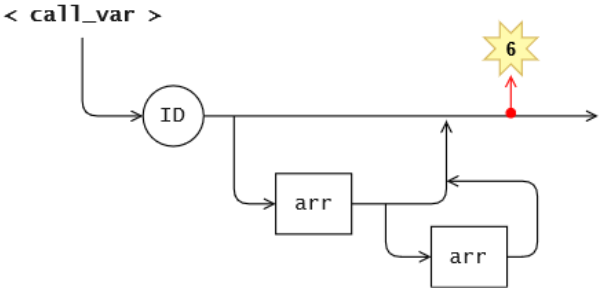
3. Guardar el tipo leído.

4. Valida que la variable no exista dentro del scope actual. Current_Var = ID. Generar la siguiente dirección virtual dependiendo del tipo. Guardar variable, su tipo y dirección en la tabla de variables del scope actual.

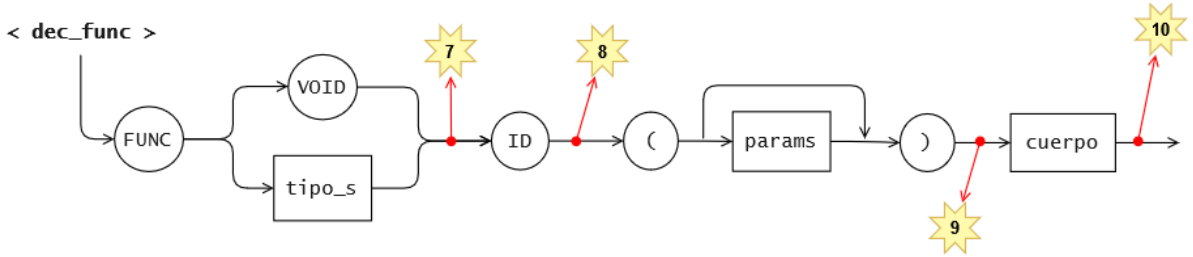


5. Agregar dimensión en current_var



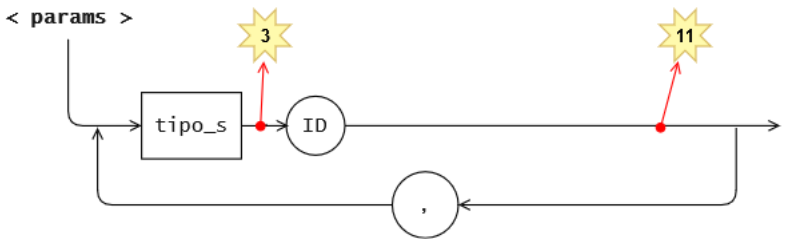


6. Revisar si la variable existe. Si existe, obtener el tipo y dirección virtual de la variable. (current_type y addr).

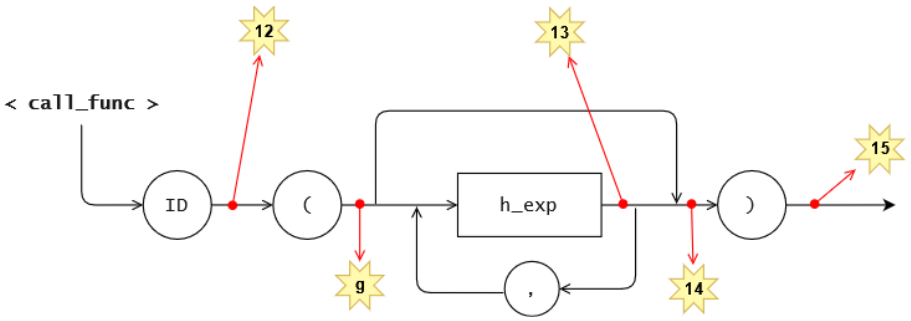


7. Guardar el tipo de la función declarada. function_type = token

8. Revisa que no exista una función con el mismo nombre. Si no, guarda el nombre y tipo de función en el directorio. Scope = ID. Si la función tiene tipo retorno, crear una variable en la tabla de globales con el nombre de la función.
9. Guardar el quad pointer en la dirección de inicio de la función actual.
10. Crear cuádruplo 'ENDFUNC' y borrar la tabla de variables del *scope* actual.



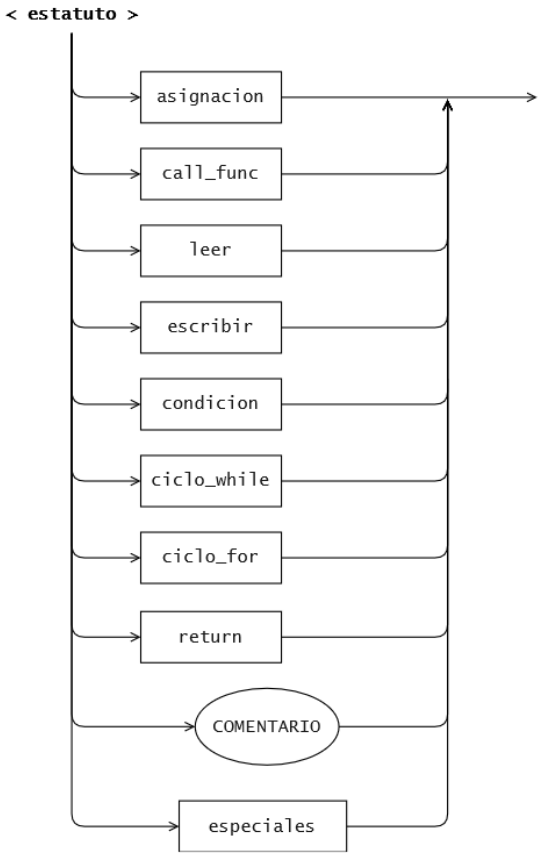
3. Guardar el tipo leído.
11. Agregar tipo de parámetro en la función actual (en el directorio de funciones). Agregar parámetro (y su dirección generada) como variable en la tabla de variables del *scope* actual.

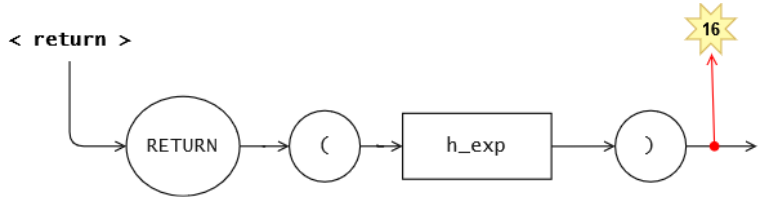


12. Verificar que la función exista. Si existe, crear cuádruplo ERA, ID. Contador de parámetros = 0

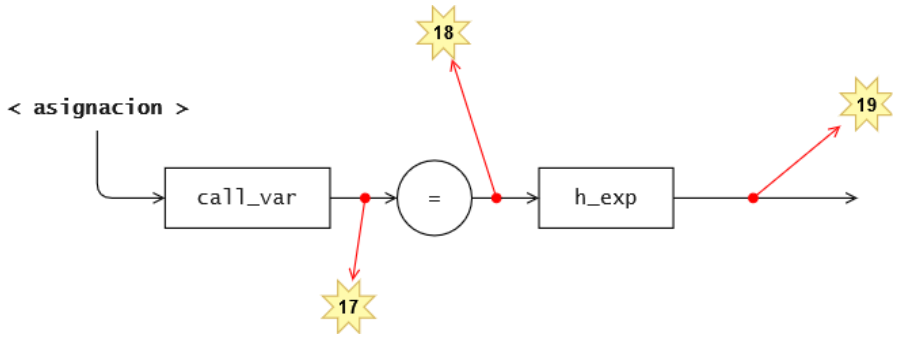
g. Añadir fondo falso a la pila de operandos. Push de '('

13. Hacer pop de la pila de operandos y tipos (arg y arg_type). Revisar que haya concordancia entre el arg_type y el número de parámetro en el que va. Si no hay error, crear cuádruplo PARAM, arg, p#param_counter, param_type.
14. Verificar que no se esperen más parámetros (que el contador de parámetros apunte a None) y eliminar fondo falso. (Pop del '(' en la pila de operandos).
15. Crear cuádruplo GOSUB, ID. Si la función tiene tipo de retorno, generar un cuádruplo =, dirección de la variable global con nombre de la función, dirección de siguiente temporal del tipo de la función.



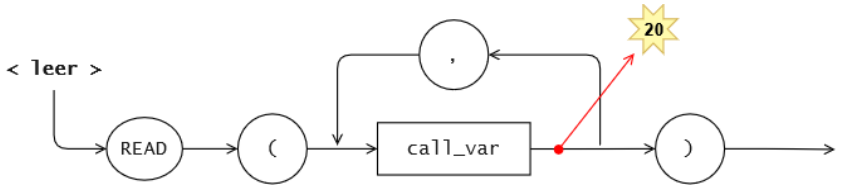


16. Revisar que la función actual (scope) no sea de tipo void.
    Hacer pop en la pila de operandos y de tipos (result y result_type). Validar que el tipo del resultado sea el mismo al de la función. Crear cuádruplo RETURN, result, dirección de la variable global con nombre de la función, .

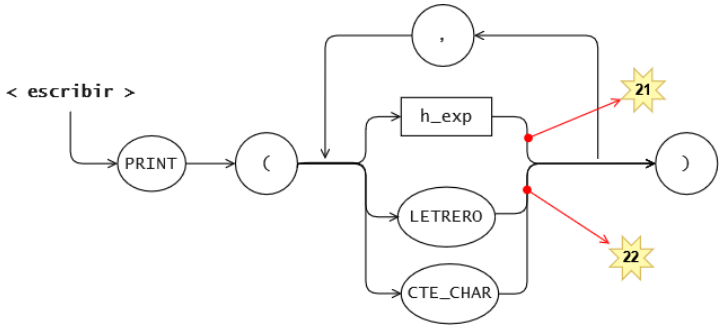


17. Meter nombre de variable y su tipo en pila de operandos y pila de tipos respectivamente.
18. Meter operador leído en pila de operadores.

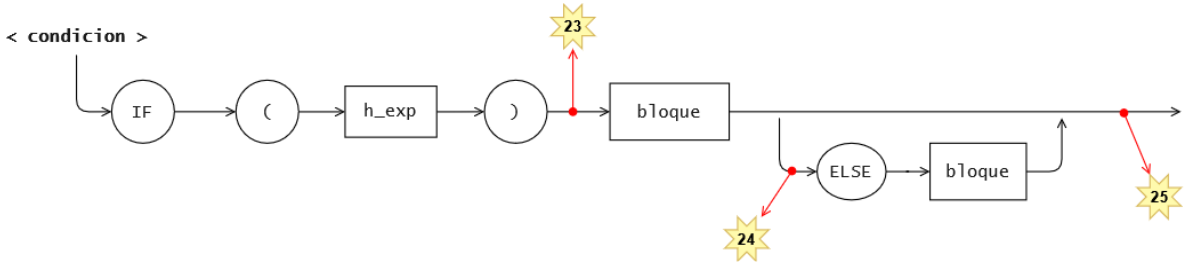
19. Hacer pop de la pila de operadores (operador), y dos pops a la pila de operandos y de tipos, (oper_Izq y oper_Der en ese orden). Validar los tipos con el operador en el cubo semántico. Si no hay error, crear cuádruplo: =, oper_Izq, , oper_Der



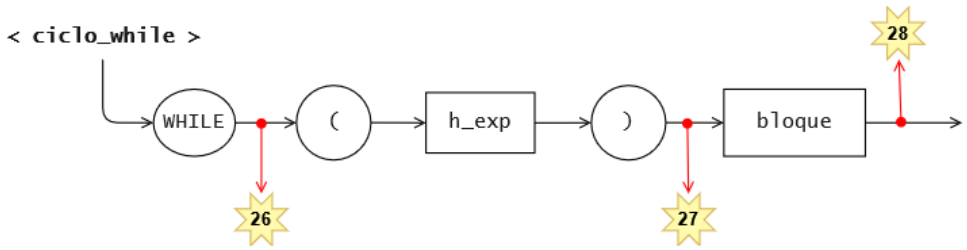
20. Crear cuádruplo READ, , token leído.



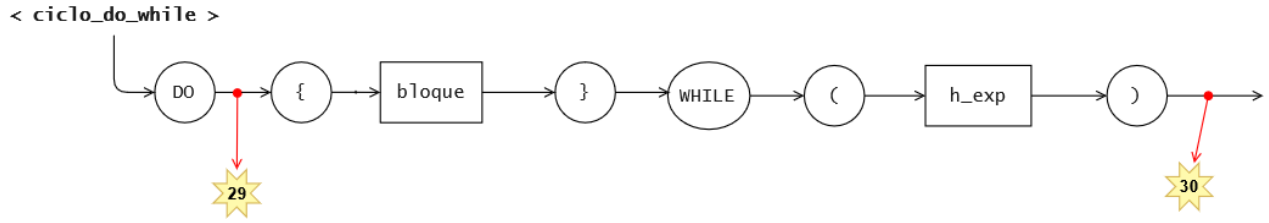
21. Hacer pop de la pila de operandos y pila de tipos. Crear cuádruplo PRINT, , resultado.
22. Crear cuádruplo PRINT, , , token leído.



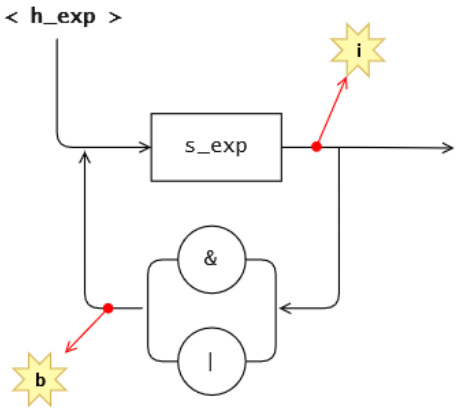
23. Hacer pop en la pila de operandos (resultado) y en la pila de tipos. Si el tipo del resultado no es booleano, mostrará error. Si es booleano, crea el cuádruplo GOTOF, resultado. Meter quad_pointer - 1 a la pila de saltos.
24. Crear cuádruplo GOTO, . Hacer pop de la pila de saltos, ir al número de cuádruplo que indica y poner en el resultado el quad_pointer. Meter quad_pointer - 1 a la pila de saltos.
25. Hacer pop de la pila de saltos, ir al número de cuádruplo que indica y poner en el resultado el quad_pointer.



26. Guardar contador de cuádruplos para saber donde volver a evaluar la expresión (Meter quad_pointer a la pila de saltos).
27. Hacer pop de la pila de operandos y pila de tipos (resultado), evaluar que el tipo del resultado sea booleano. Si es, crear cuádruplo GOTOF, result, __. Meter quad_pointer a la pila de saltos.
28. Hacer dos pops en la pila de saltos: quad_incompleto y return. Crear cuádruplo GOTO, , return. Ir al número de cuádruplo de quad_incompleto y poner en resultado quad_pointer.

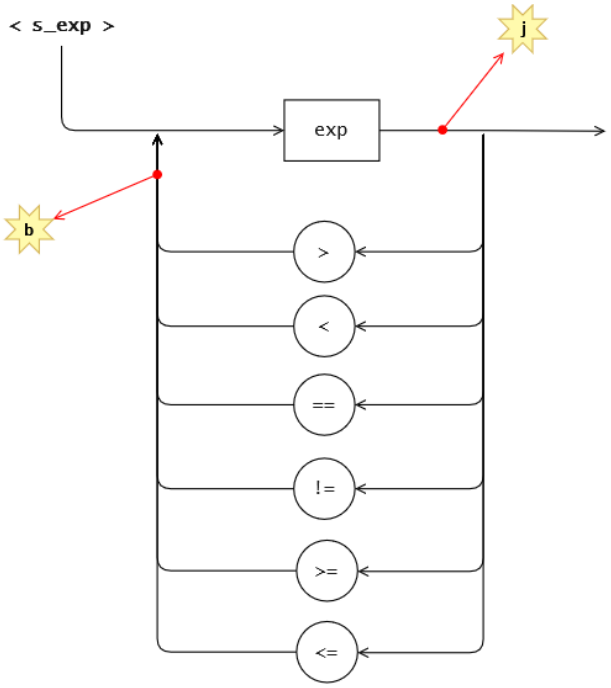


29. Guardar contador de cuádruplos para saber donde volver a hacer el ciclo. (Meter quad_pointer a la pila de saltos).
30. Hacer pop de la pila de operandos y pila de tipos (resultado), evaluar que el tipo del resultado sea booleano. Si es, crear cuádruplo GOTOV, result, pop a la pila de saltos.



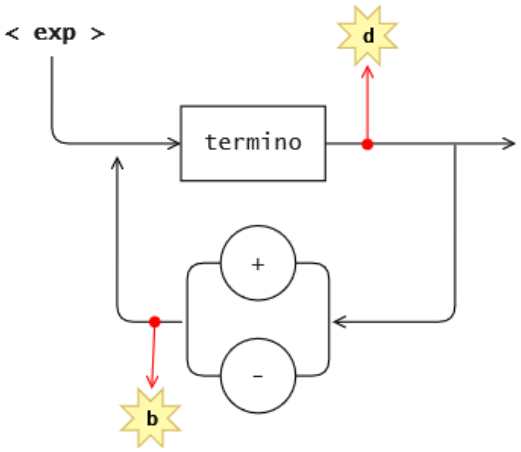
i. Si el top de la pila de operadores es '&' o '|' generar cuádruplo correspondiente y en resultado poner un nuevo temporal. Hacer push del temporal y su tipo a las pilas.

b. Guardar el operador en la pila de operadores.



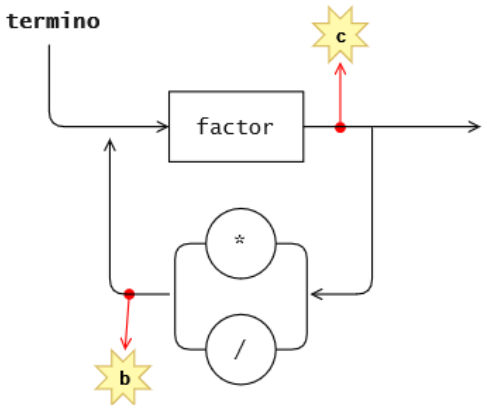
j. Si el top de la pila de operadores es ‘>’, ‘<’, ‘==’, ‘!=’, ‘>=’ o ‘<=’’ generar cuádruplo correspondiente y en resultado poner un nuevo temporal. Hacer push del temporal y su tipo a las pilas.

b. Guardar el operador en la pila de operadores.

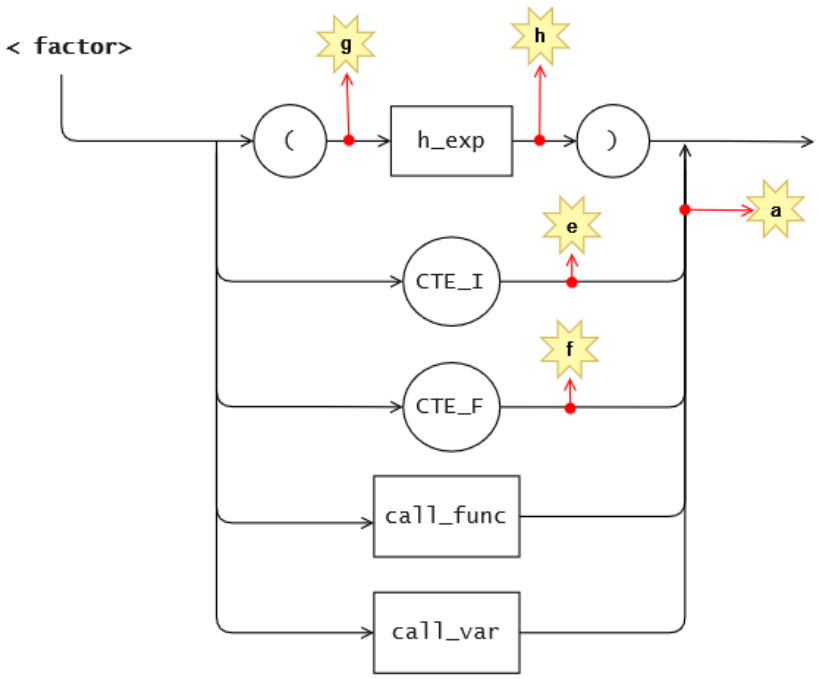


d. Si el top de la pila de operadores es ‘+’ o ‘-’ generar cuádruplo correspondiente y en resultado poner un nuevo temporal. Hacer push del temporal y su tipo a las pilas.

b. Guardar el operador en la pila de operadores.



d. Si el top de la pila de operadores es ‘*’ o ‘/’ generar cuádruplo correspondiente y en resultado poner un nuevo temporal. Hacer push del temporal y su tipo a las pilas.
b. Guardar el operador en la pila de operadores.



a. Meter el token leído y su tipo en las pilas correspondientes.
e. current_type = int. Si el valor está en la tabla de constantes, obtener la dirección asignada. Si no, agregarlo a la tabla de constantes.
f. current_type = float. Si el valor está en la tabla de constantes, obtener la dirección asignada. Si no, agregarlo a la tabla de constantes.
g. Añadir ‘p’ a la pila de Operandos.
h. Si hay ‘(‘ en la pila de operandos, hacer pop en la pila de Operandos. Si no, mostrar error.

3.4.2 Códigos de operación

Código	Lado Izq	Lado Der	Resultado	Acción a realizar
GOTO			quad	Salto incondicional
GOTOF	result		quad	Salto si el resultado es falso.
GOTOT	result		quad	
PRINT			result	Imprime lo que se encuentra en result.
READ			variable	Lee input del usuario y lo guarda en la dirección de variable.
=	valor		variable	Guardar el valor en la dirección de la variable.
+	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, sumarlos y dejar el resultado en la dirección de result.
-	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, restarlos y dejar el resultado en la dirección de result.
*	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, multiplicarlos y dejar el resultado en la dirección de result.
/	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, dividirlos y dejar el resultado en la dirección de result.
>	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq > Der y dejar el resultado en la dirección de result.
<	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq < Der y dejar el resultado en la dirección de result.
>=	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq >= Der y dejar el resultado en la dirección de result.
<=	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq <= Der y

				dejar el resultado en la dirección de result.
==	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq == Der y dejar el resultado en la dirección de result.
!=	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq != Der y dejar el resultado en la dirección de result.
&	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq and Der y dejar el resultado en la dirección de result.
|	Lado Izq	Lado Der	result	Obtener los valores del lado derecho e izquierdo, comparar Izq or Der y dejar el resultado en la dirección de result.
ERA	función			Conseguir los recursos de la función del directorio de funciones y crear memoria.
PARAM	valor	dirección	tipo	Ir a la memoria de la función, agregar el valor en la dirección dada en el bloque del tipo correspondiente.
GOSUB	función			Dormir la memoria actual y guardar el apuntador (IP) a la instrucción actual. Hacer cambio de contexto a la nueva memoria y apuntar el IP a la dirección de inicio de la función.
RETURN	valor		dirección de variable global	Asignar el valor de retorno a la variable global que tiene el mismo nombre que la función.
ENDFUNC				Regresar al contexto anterior: despertar la memoria en la pila de ejecución y actualizar IP al guardado.

3.4.3 Direcciones virtuales

Las direcciones se asignan por orden de llegada (declaración de variables o generación de temporales) en base a los siguientes rangos iniciales:

```
# ------ Globals -------- # 4 digits
GLOBAL_INT_START = 1000
GLOBAL_INT_END = 2500

GLOBAL_FLOAT_START = 3000
GLOBAL_FLOAT_END = 4500

GLOBAL_CHAR_START = 5000
GLOBAL_CHAR_END = 6500

GLOBAL_BOOL_START = 7000
GLOBAL_BOOL_END = 8500

# ------ Locals  -------- # 5 digits
LOCAL_INT_START = 10000
LOCAL_INT_TEMP_START = 13000
LOCAL_INT_END = 15000

LOCAL_FLOAT_START = 30000
LOCAL_FLOAT_TEMP_START = 33000
LOCAL_FLOAT_END = 35000

LOCAL_CHAR_START = 50000
LOCAL_CHAR_TEMP_START = 53000
LOCAL_CHAR_END = 55000

LOCAL_BOOL_START = 70000
LOCAL_BOOL_TEMP_START = 73000
LOCAL_BOOL_END = 75000

# ------ Constants -------- # 6 digits
CONST_INT_START = 100000
CONST_INT_END = 110000

CONST_FLOAT_START = 300000
CONST_FLOAT_END = 310000

CONST_CHAR_START = 500000
```

```
CONST_CHAR_END = 510000

CONST_BOOL_START = 700000
CONST_BOOL_END = 710000
```

3.4.4 Tabla de consideraciones semánticas

Si no se genera un tipo de resultado, se maneja como un error.

Lado Izq	Lado Der	+, -	*	/	Relacionales >, <, >=, <=	Igualdad ==, !=	Lógicos &, |
int	int	int	int	int	bool	bool	x
int	float	float	float	int	bool	bool	x
float	int	float	float	int	bool	bool	x
float	float	float	float	float	bool	bool	x
bool	bool	x	x	x	x	bool	bool
char	char	x	x	x	x	bool	x

3.5 Arquitectura de memoria en compilación

3.5.1 Directorio de funciones

Se utilizó una clase “Functions_Directory.py” que contuviera un diccionario para guardar las funciones declaradas. La estructura del diccionario es:

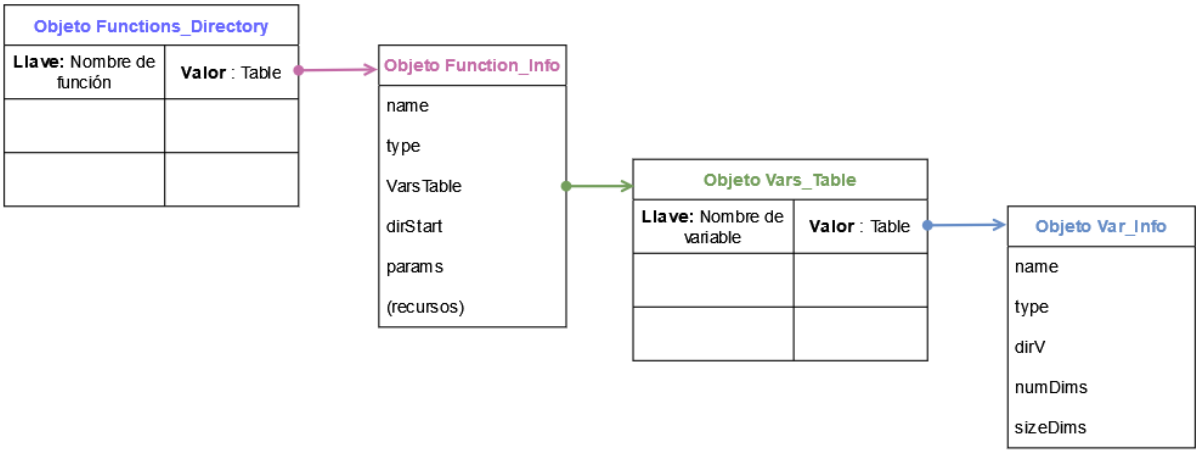
```
{ (llave) Nombre de la función : (valor) Objeto Function_Info }
```

Esto con el objetivo de que el nombre de la función, al ser la llave, sea única y la búsqueda del nombre esté optimizada (como funcionan por default los diccionarios)

En el objeto de Function_Info se guarda en los atributos la información de la función:

- name: Nombre de la función
- type: Tipo de la función
- varsTable: Objeto que guarda la tabla de variables.
- dirStart: Dirección del cuádruplo donde inicia la función.
- params: Lista de los tipos de los parámetros
- 8 atributos para el conteo de variables locales y temporales de enteros, flotantes, char y bool.

Gráfico de la relación entre directorio de funciones y tabla de variables



3.5.2 Tabla de Variables

Similar a las funciones. Se creó una clase “Functions_Directory.py” que contuviera un diccionario para guardar las funciones declaradas. La estructura del diccionario es:

```
{ (llave) Nombre de la variable : (valor) Objeto Var_Info }
```

Los diccionarios están optimizados para búsqueda por llave, la cual debe ser única, por lo que estas características son ideales para guardar y buscar nombres de variables.

En el objeto de Var_Info se guarda en los atributos la información de la variable:

- name: Nombre de la variable
- type: Tipo de la variable
- dirV: Dirección virtual asignada a la variable
- numDimensions: Número de dimensiones que tiene la variable
- sizeDimensions: Lista que guarda los tamaños de cada dimensión de la variable.

3.5.3 Tabla de Constantes

Diccionario que guarda las constantes y su dirección virtual dentro del rango definido anteriormente y es asignada por orden de llegada. La estructura del diccionario es:

```
{ (llave) Valor de la constante : (valor) Dirección Virtual }
```

La llave es el valor de la constante porque se prioriza la búsqueda por valor, para saber si una constante ya está dada de alta en la tabla.

3.5.4 Cuádruplos

Es un objeto que guarda la estructura de los cuádruplos, con los métodos de obtener y guardar en la casilla de resultado. En los atributos se guardan los siguientes datos:

operador	Lado Izquierdo	Lado Derecho	Resultado

Y tiene los métodos get_Result() y set_Result(result) para poder obtener o modificar el valor que se encuentra en la casilla de resultado.

Se creó una lista para ir guardando los cuádruplos, añadiendo un 'None' al inicio para que comience la numeración a partir de 1.

3.5.5 Pilas

Se implementó usando la librería de collections.deque ya que tiene mejor optimización que las listas para hacer operaciones de *append* y *pop* (tiempo de O(1)).

Ya que deque puede ser utilizado por el lado izquierdo y derecho, para simular un stack se utilizaron las operaciones de append y pop, para agregar u obtener elementos al final de la lista, recreando el comportamiento *Last in - First out*.

Se usaron pilas para:

- ➢ Operandos
- ➢ Operadores
- ➢ Tipos
- ➢ Saltos

4 Acerca de la máquina virtual

4.1 Lenguaje y librerías usadas

El compilador fue realizado en un equipo con sistema operativo Windows. Se desarrolló en Python con las siguientes librerías:

- deque de la librería collections: Para la implementación de pilas.
- pandas: Para pasar los cuádruplos a formato de *dataframe*.

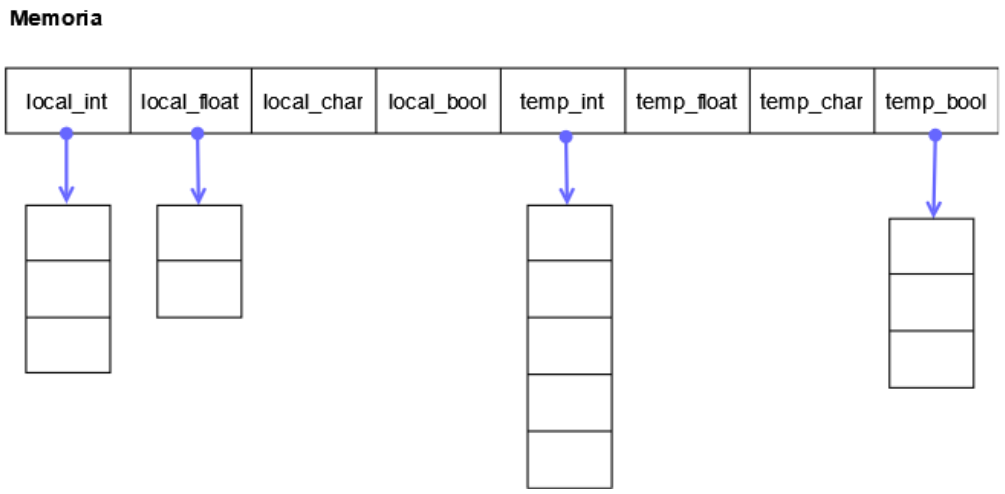
- numpy: Para hacer uso de enteros de 64 bits.

4.2 Arquitectura de memoria en ejecución

Para la simulación de memoria se creó una clase que recibe el número de recursos y genera arreglos del tamaño recibido. Separa los tipos locales de los tipos temporales, para tener mejor organización en base al rango de direcciones.

```
class Memory:
      def __init__(self, li, lf, lc, lb, ti, tf, tc, tb):
      self.local_int = np.zeros(li, dtype=np.int64)
      self.local_float = np.zeros(lf, dtype=np.float64)
      self.local_char = np.zeros(lc, dtype="U1")
      self.local_bool = np.zeros(lb, dtype=bool)

      self.temp_int = np.zeros(ti, dtype=np.int64)
      self.temp_float = np.zeros(tf, dtype=np.float64)
      self.temp_char = np.zeros(tc, dtype="U1")
      self.temp_bool = np.zeros(tb, dtype=bool)
```

Representación gráfica de la memoria:



4.2.1 Memoria local

Cada que hay un nuevo contexto se genera un espacio de memoria local (con ERA se crea y con GOSUB se activa, es decir, se vuelve la memoria actual).

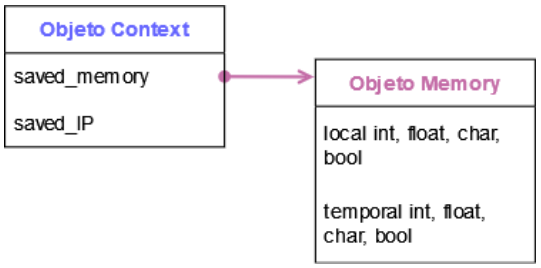
4.2.2 Memoria global

Es una memoria que tiene la misma estructura que la local (de la clase memoria) pero no se borra, se mantiene presente en un atributo de la clase Virtual_Machine, para tenerla accesible.

Las variables con el nombre local_x vendrían a ser el espacio de las variables globales.

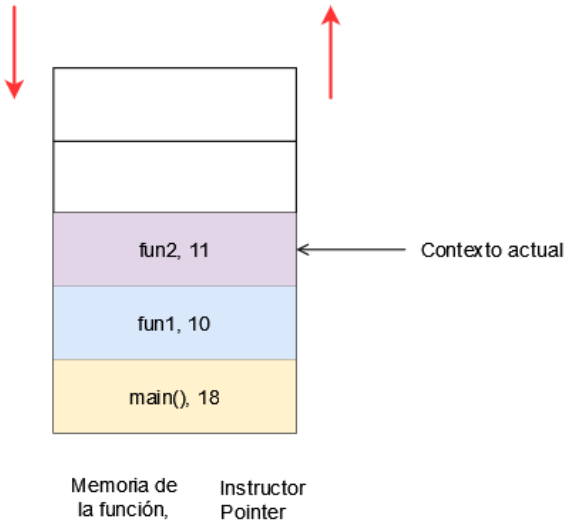
4.2.3 Contexto

Es una clase en donde se guarda un objeto memoria y un instruction pointer, para poder guardar el contexto actual antes de hacer cambio a una llamada de función, donde se creará una nueva memoria y el instructor_pointer se moverá para la ejecución de los cuádruplos de la función. Sin embargo, es importante guardar el contexto donde se hace la llamada para poder regresar una vez terminada la función.



Los contextos se van guardando en una pila para tener el comportamiento Last In - First Out, pudiendo regresar al contexto más “nuevo” de la pila, lo que vendría siendo el contexto en el que se estaba antes de la llamada a función.

Ejemplo: Se ejecuta un programa que en el cuadruplo 18 llama a una función fun1 (cuadruplo 4) y dentro de dicha función en el cuadruplo 10, se llama la función fun2. El stack de contextos se vería de la siguiente manera:

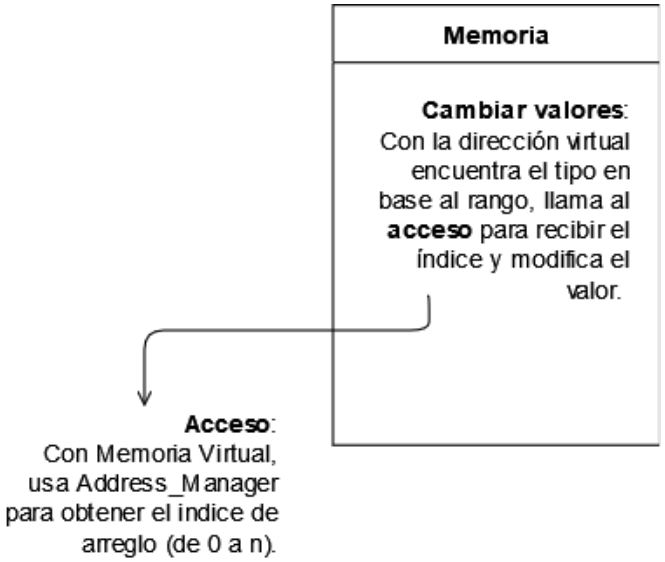


4.3 Asociación entre direcciones virtuales y de ejecución

Ver sección 3.4.3 Direcciones virtuales para la declaración de rangos de acuerdo al tipo de variable y su scope (global, local, temporal o constante).

Se maneja la clase 'Address_Manager.py' para asignar las direcciones durante la compilación. Además, esta clase tiene métodos como:

- get_Memory_Index (virtual_addr) : Recibe una dirección virtual y la transforma a índice de arreglo (empezando de 0 a n, el número de recursos).



5 Pruebas del funcionamiento

5.1 Factorial recursivo

5.1.1 Código

```
program Factorial:

func int factorial(int x)
{
      #print("Inside factorial with x =", x)
      if (x == 1)
      {
      return (x)
      }
      else
      {
      return (x * factorial(x - 1))
```

```
        }
}

main()
{
        var int a
        read(a)
        print("Factorial: ", factorial(a))
}
```

5.1.2 Código intermedio generado

	Operator	Left O.	Right O.	Result
1	GOTO			14
2	==	10000	100000	73000
3	GOTOF	73000		6
4	RETURN	10000		1000
5	GOTO			13
6	ERA	factorial		
7	-	10000	100000	13000
8	PARAM	13000	p#0	int
9	GOSUB	factorial		
10	=	1000		13001
11	*	10000	13001	13002
12	RETURN	13002		1000
13	ENDFUNC			
14	READ			1001
15	PRINT			"Factorial: "
16	ERA	factorial		
17	PARAM	1001	p#0	int
18	GOSUB	factorial		
19	=	1000		13000
20	PRINT			13000

5.1.3 Resultados de ejecución

```
C:\Users\Jazyd\Documents\Escuela\Diseño de Compiladores> python
TapiCompi/compiler/TapiCompi.py TapiCompi/Examples/Factorial.tapi
6
Factorial:
720
```

6 Documentación del código del proyecto

6.1 TapiCompi.py

La compilación se manda a llamar ejecutando el archivo 'TapiCompi.py' y se envía como argumento el nombre del archivo que se quiere ejecutar. Por ejemplo, para ejecutar el archivo Factorial.tapi tendría que usarse la siguiente instrucción (desde la carpeta compiler):

```
python TapiCompi.py Examples/Factorial.tapi
```

Descripción: Archivo que maneja el flujo de compilación del lenguaje TapiCompi. Manda a llamar al *parser* y a la máquina virtual. Si se ejecuta el comando vm.run con un True entonces se mostratán los elementos de debug (directorio de funciones, tabla de constantes y cuadruplos generados).

Entrada: El nombre del archivo a ejecutar.

Salida: Muestra en consola las instrucciones

Implementación:

```
import sys
from src.TapiCompi_pars import parser, quadruples, directory,
const_table
from src.TapiCompi_lex import Tokenize
from vm.Virtual_Machine import Virtual_Machine

def run(code):
      #Tokenize(code)
      try:
      #Tokenize(data)
      if(parser.parse(data) == "Success"):
            #print('\nCompilacion exitosa')
            vm = Virtual_Machine(quadruples, directory, const_table)
            vm.run(True)


      else:
            print('\nCompilacion fallida')
            return

      except EOFError:
```

```
        print(EOFError)


if __name__ == '__main__':
        if len(sys.argv) > 1:
        #print(sys.argv[1])
        data = open(sys.argv[1]).read()
        run(data)
        else:
        print("Error: No se ha ingresado un archivo de entrada.")
```

6.2 CuboSem.py

Descripción: Clase que contiene un diccionario de diccionarios de diccionarios, permitiendo tener de esta forma la estructura de Cubo_Sem['==']['bool']['bool']

Entrada: Operador, Tipo Izquierdo, Tipo Derecho.

Donde se utiliza: En el parser.

Salida: Regresa el tipo de resultado de una operación en base a los tipos de los operandos. En caso de no ser posible una combinación, regresa -1.

Implementación:

```
import numpy as np
import pandas as pd

dict_Mult = {
        'int' : {'int': 'int', 'float': 'float'},
        'float': {'int': 'float', 'float': 'float'},
}

dict_Div = {
        'int' : {'int': 'int', 'float': 'int'},
        'float': {'int': 'int', 'float': 'float'},
}

dict_SumRes = {
        'int': {'int': 'int', 'float': 'float'},
        'float': {'int': 'float', 'float': 'float'},
```

```
}

dict_Relacionales = {
      'int': {'int': 'bool', 'float': 'bool'},
      'float': {'int': 'bool', 'float': 'bool'},
}

dict_EqDes = dict_Relacionales.copy()
dict_EqDes['bool'] = {'bool': 'bool'}
dict_EqDes['char'] = {'char': 'bool'}

dict_Logicos = {
      'bool': {'bool': 'bool'},
}

dict_Assign = {
      'int': {'int': 'int', 'float': 'int'},
      'float': {'float': 'float', 'int': 'float'},
      'bool': {'bool': 'bool'},
      'char': {'char': 'char'}
}

# Create semantic cube for operations between types of variables

Cubo_Sem = {
      # Arithmetic operators
      '*': dict_Mult,
      '/': dict_Div,
      '+': dict_SumRes,
      '-': dict_SumRes,

      # Relational operators
      '>': dict_Relacionales,
      '<': dict_Relacionales,
      '>=': dict_Relacionales,
      '<=': dict_Relacionales,
      '==': dict_EqDes,
      '!=': dict_EqDes,

      # Logic operators
      '&': dict_Logicos,
      '|': dict_Logicos,
```

```
        # Asssign
        '=': dict_Assign
}

df_cuboSem = pd.DataFrame(Cubo_Sem)

# Access the type_res of an expression
# df_cuboSem['==']['bool']['bool']

def validate_type(
        operator,
        typeL : str,
        typeR : str) -> str:
        '''
        Function that checks if the operation between two types is
valid.

        Args:
        operator (char): Operator of the operation.
        typeL (Type): Type of the left operand.
        typeR (Type): Type of the right operand.

        Returns:
        Type: Type of the result of the operation.
        -1 if the operation is not valid.
        '''
        try:
        type_res = df_cuboSem[operator][typeL][typeR]
        return type_res
        except:
        return -1 # Error code to handle mismatched types
```

6.3 Address_Manager.py

Descripción: Esta clase genera direcciones virtuales que asignar a las variables, dependiendo de su tipo y en qué scope estará (global, local, temporal, constante)

Entrada y salida: tipo (para obtener la siguiente dirección disponible) o memoria virtual (para obtener el índice del arreglo en base al rango definido).

Donde se utiliza: En el parser y máquina virtual.

Implementación:

```
from libs.utils.Constants import *

# Class to map the variables to a virtual direction

# Each pointer points to the next available virtual direction
class Address_Manager:
      def __init__(self):
      # Global variables
      self.global_int_pointer = GLOBAL_INT_START
      self.global_float_pointer = GLOBAL_FLOAT_START
      self.global_char_pointer = GLOBAL_CHAR_START
      self.global_bool_pointer = GLOBAL_BOOL_START

      # Local & temporal variables
      self.local_int_pointer = LOCAL_INT_START
      self.local_int_temp_pointer = LOCAL_INT_TEMP_START

      self.local_float_pointer = LOCAL_FLOAT_START
      self.local_float_temp_pointer = LOCAL_FLOAT_TEMP_START

      self.local_char_pointer = LOCAL_CHAR_START
      self.local_char_temp_pointer = LOCAL_CHAR_TEMP_START

      self.local_bool_pointer = LOCAL_BOOL_START
      self.local_bool_temp_pointer = LOCAL_BOOL_TEMP_START

      # Constant variables
      self.const_int_pointer = CONST_INT_START
      self.const_float_pointer = CONST_FLOAT_START
      self.const_char_pointer = CONST_CHAR_START
      self.const_bool_pointer = CONST_BOOL_START

      #Getters for new directions
      def get_Global_Int_Dir(self):
      if (self.global_int_pointer > GLOBAL_INT_END):
            print("Error: Global Int pointer out of range")
            return -1
      else:
            self.global_int_pointer += 1
            return self.global_int_pointer - 1
```

```
def get_Global_Float_Dir(self):
if (self.global_float_pointer > GLOBAL_FLOAT_END):
      print("Error: Global Float pointer out of range.")
      return -1
else:
      self.global_float_pointer += 1
      return self.global_float_pointer - 1

def get_Global_Char_Dir(self):
if (self.global_char_pointer > GLOBAL_CHAR_END):
      print("Error: Global Char pointer out of range.")
      return -1
else:
      self.global_char_pointer += 1
      return self.global_char_pointer - 1

def get_Global_Bool_Dir(self):
if (self.global_bool_pointer > GLOBAL_BOOL_END):
      print("Error: Global Bool pointer out of range.")
      return -1
else:
      self.global_bool_pointer += 1
      return self.global_bool_pointer - 1

def get_Local_Int_Dir(self):
if (self.local_int_pointer >= LOCAL_INT_TEMP_START):
      print("Error: Local Int pointer out of range.")
      return -1
else:
      self.local_int_pointer += 1
      return self.local_int_pointer - 1

def get_Local_Int_Temp_Dir(self):
if (self.local_int_temp_pointer > LOCAL_INT_END):
      print("Error: Local Int Temp pointer out of range.")
      return -1
else:
      self.local_int_temp_pointer += 1
      return self.local_int_temp_pointer - 1

def get_Local_Float_Dir(self):
```

```
    if (self.local_float_pointer >= LOCAL_FLOAT_TEMP_START):
          print("Error: Local Float pointer out of range.")
          return -1
    else:
          self.local_float_pointer += 1
          return self.local_float_pointer - 1

    def get_Local_Float_Temp_Dir(self):
    if (self.local_float_temp_pointer > LOCAL_FLOAT_END):
          print("Error: Local Float Temp pointer out of range.")
          return -1
    else:
          self.local_float_temp_pointer += 1
          return self.local_float_temp_pointer - 1

    def get_Local_Char_Dir(self):
    if (self.local_char_pointer >= LOCAL_CHAR_TEMP_START):
          print("Error: Local Char pointer out of range.")
          return -1
    else:
          self.local_char_pointer += 1
          return self.local_char_pointer - 1

    def get_Local_Char_Temp_Dir(self):
    if (self.local_char_temp_pointer > LOCAL_CHAR_END):
          print("Error: Local Char Temp pointer out of range.")
          return -1
    else:
          self.local_char_temp_pointer += 1
          return self.local_char_temp_pointer - 1


    def get_Local_Bool_Dir(self):
    if (self.local_bool_pointer > LOCAL_BOOL_TEMP_START):
          print("Error: Local Bool pointer out of range.")
          return -1
    else:
          self.local_bool_pointer += 1
          return self.local_bool_pointer - 1

    def get_Local_Bool_Temp_Dir(self):
    if (self.local_bool_temp_pointer > LOCAL_BOOL_END):
```

```
            print("Error: Local Bool Temp pointer out of range.")
            return -1
      else:
            self.local_bool_temp_pointer += 1
            return self.local_bool_temp_pointer - 1

      def get_Const_Int_Dir(self):
      if (self.const_int_pointer > CONST_INT_END):
            print("Error: Const Int pointer out of range.")
            return -1
      else:
            self.const_int_pointer += 1
            return self.const_int_pointer - 1

      def get_Const_Float_Dir(self):
      if (self.const_float_pointer > CONST_FLOAT_END):
            print("Error: Const Float pointer out of range.")
            return -1
      else:
            self.const_float_pointer += 1
            return self.const_float_pointer - 1

      def get_Const_Char_Dir(self):
      if (self.const_char_pointer > CONST_CHAR_END):
            print("Error: Const Char pointer out of range.")
            return -1
      else:
            self.const_char_pointer += 1
            return self.const_char_pointer - 1

      def get_Const_Bool_Dir(self):
      if (self.const_bool_pointer > CONST_BOOL_END):
            print("Error: Const Bool pointer out of range.")
            return -1
      else:
            self.const_bool_pointer += 1
            return self.const_bool_pointer - 1

      def get_Global_Dir(self,  type):
      match type:
            case "int":
                  return self.get_Global_Int_Dir()
```

```
            case "float":
                  return self.get_Global_Float_Dir()
            case "char":
                  return self.get_Global_Char_Dir()
            case "bool":
                  return self.get_Global_Bool_Dir()

      def get_Local_Dir(self,  type):
      match type:
            case "int":
                  return self.get_Local_Int_Dir()
            case "float":
                  return self.get_Local_Float_Dir()
            case "char":
                  return self.get_Local_Char_Dir()
            case "bool":
                  return self.get_Local_Bool_Dir()


      def get_Local_Temporal_Dir(self, type):
      match type:
            case "int":
                  return self.get_Local_Int_Temp_Dir()
            case "float":
                  return self.get_Local_Float_Temp_Dir()
            case "char":
                  return self.get_Local_Char_Temp_Dir()
            case "bool":
                  return self.get_Local_Bool_Temp_Dir()

      def reset_local(self):
      # Local & temporal variables
      self.local_int_pointer = LOCAL_INT_START
      self.local_int_temp_pointer = LOCAL_INT_TEMP_START

      self.local_float_pointer = LOCAL_FLOAT_START
      self.local_float_temp_pointer = LOCAL_FLOAT_TEMP_START

      self.local_char_pointer = LOCAL_CHAR_START
      self.local_char_temp_pointer = LOCAL_CHAR_TEMP_START

      self.local_bool_pointer = LOCAL_BOOL_START
```

```
      self.local_bool_temp_pointer = LOCAL_BOOL_TEMP_START

# Conversion of virtual address to memory array index
def get_Memory_Index(virtual_addr):
      if (virtual_addr >= GLOBAL_INT_START and virtual_addr <=
GLOBAL_INT_END):
      return virtual_addr - GLOBAL_INT_START
      elif (virtual_addr >= GLOBAL_FLOAT_START and virtual_addr <=
GLOBAL_FLOAT_END):
      return virtual_addr - GLOBAL_FLOAT_START
      elif (virtual_addr >= GLOBAL_CHAR_START and virtual_addr <=
GLOBAL_CHAR_END):
      return virtual_addr - GLOBAL_CHAR_START
      elif (virtual_addr >= GLOBAL_BOOL_START and virtual_addr <=
GLOBAL_BOOL_END):
      return virtual_addr - GLOBAL_BOOL_START
      elif (virtual_addr >= LOCAL_INT_START and virtual_addr <
LOCAL_INT_TEMP_START):
      return virtual_addr - LOCAL_INT_START
      elif (virtual_addr >= LOCAL_INT_TEMP_START and virtual_addr
<= LOCAL_INT_END):
      return virtual_addr - LOCAL_INT_TEMP_START
      elif (virtual_addr >= LOCAL_FLOAT_START and virtual_addr <
LOCAL_FLOAT_TEMP_START):
      return virtual_addr - LOCAL_FLOAT_START
      elif (virtual_addr >= LOCAL_FLOAT_TEMP_START and virtual_addr
<= LOCAL_FLOAT_END):
      return virtual_addr - LOCAL_FLOAT_TEMP_START
      elif (virtual_addr >= LOCAL_CHAR_START and virtual_addr <
LOCAL_CHAR_TEMP_START):
      return virtual_addr - LOCAL_CHAR_START
      elif (virtual_addr >= LOCAL_CHAR_TEMP_START and virtual_addr
<= LOCAL_CHAR_END):
      return virtual_addr - LOCAL_CHAR_TEMP_START
      elif (virtual_addr >= LOCAL_BOOL_START and virtual_addr <
LOCAL_BOOL_TEMP_START):
      return virtual_addr - LOCAL_BOOL_START
      elif (virtual_addr >= LOCAL_BOOL_TEMP_START and virtual_addr
<= LOCAL_BOOL_END):
      return virtual_addr - LOCAL_BOOL_TEMP_START
      elif (virtual_addr >= CONST_INT_START and virtual_addr <=
CONST_INT_END):
```

```
      return virtual_addr - CONST_INT_START
      elif (virtual_addr >= CONST_FLOAT_START and virtual_addr <=
CONST_FLOAT_END):
      return virtual_addr - CONST_FLOAT_START
      elif (virtual_addr >= CONST_CHAR_START and virtual_addr <=
CONST_CHAR_END):
      return virtual_addr - CONST_CHAR_START
      elif (virtual_addr >= CONST_BOOL_START and virtual_addr <=
CONST_BOOL_END):
      return virtual_addr - CONST_BOOL_START
      else:
      print("Error: Virtual address out of range.")
      return -1
```

7 Manual de usuario

7.1 Quick Reference Manual

Se puede revisar el manual de usuario en el [readme del repositorio en github]

7.2 Video Demo

Video en youtube.

8 Anexos

Anexo 1. Historial de commits

* 3f1ccce - 2022-11-21 : Deleted unused tokens (for statement not developed)
* 84bfe72 - 2022-11-21 : Deleted simicolon for var declarations
* e7e18ed - 2022-11-21 : Fixed incorrect output
* 5ac480c - 2022-11-21 : Fixed recursion issues.
* ad759ee - 2022-11-21 : Added operation code RETURN and fixed the assignation of returned value
* 6d6b0bd - 2022-11-21 : Added correct execution for operation codes related to functions (ERA, PARAM, GOSUB and ENDFUNC)
* f3bbdb1 - 2022-11-21 : Added code to vm to process jumps in conditionals (GOTOF and GOTOT)
* a7bd543 - 2022-11-21 : Extracted set and get values from memory to its own function in vm. Added global and current memory pointers.
*   0f3d4de - 2022-11-19 : Merge pull request #9 from JazminSantibanez/vm

```
|\
| * 970ff78 - 2022-11-19 : Added relational and logical operations to vm
| * aba1343 - 2022-11-19 : Added substract, division and multiplication code for vm
| * e15bf41 - 2022-11-18 : Update .gitignore
| * c1c386a - 2022-11-18 : Changed address start from range + 0, added const table,  vm
process print, read, assign, add and subtr operations
| * 41e010b - 2022-11-17 : Fixed bugs (description), started memory implementation and vm
|/
*   1a23da6 - 2022-11-16 : Merge pull request #8 from JazminSantibanez/codigo-functions
|\
| * baf274d - 2022-11-16 : Fixed print of variables table
| * 29db5d0 - 2022-11-15 : Started implementation for assign of virtual address
| * 43ea67a - 2022-11-15 : Added code of return statement
| * ca0cfcd - 2022-11-15 : Added code for functions
|/
*   74eabeb - 2022-11-14 : Merge pull request #7 from JazminSantibanez/codigo-ciclos
|\
| * 38e3623 - 2022-11-14 : Added do while semantics and quadruples
| * 852455b - 2022-11-14 : Removed numParams, fixed print function directory and corrected
test cases to new syntaxis of statements
| * 58b1083 - 2022-11-08 : Changed neuralgic points names to include an "n" at the beginning
| * 859f306 - 2022-11-08 : Added code to create quadruples for while, renamed test files
|/
* 566ac73 - 2022-11-07 : Correction of error handle
*   2adf327 - 2022-11-07 : Merge pull request #6 from JazminSantibanez/Codigo-if
|\
| * a8da906 - 2022-11-07 : Added code for logic and relational operators
| * bf93193 - 2022-11-07 : Changed quads print to dataframe
| * 2051408 - 2022-11-06 : Added code for conditions
|/
*   cc71e86 - 2022-11-04 : Merge pull request #5 from JazminSantibanez/Codigo-Expresiones
|\
| * ab89d82 - 2022-11-04 : Added support for parenthesis in expressions
| * d32d64a - 2022-11-04 : Added quadruples for add, substract, multiply and divide.
| * 1bd6a73 - 2022-11-04 : Fixed Print quadruple
|/
* 3ee1618 - 2022-10-31 : Added quadruple generation for assign operation
*   a3b4dea - 2022-10-31 : Merge pull request #4 from JazminSantibanez/codigo-assign
|\
| * 1f96299 - 2022-10-31 : Changed print format for tables
|/
* ebb1336 - 2022-10-25 : Added error to manage variables and functions already declared
*   ca5740e - 2022-10-25 : Merge pull request #3 from JazminSantibanez/declaration-functions
```

```
|\
| * 0c94e88 - 2022-10-25 : Implemented registry of functions in the directory
|/
* 445966e - 2022-10-25 : Implemented saving for variables with dimensions
* c1554c3 - 2022-10-25 : Fixed bug in dec_var recursion
* 1ae408e - 2022-10-25 : Added point to create functions directory and global variable table.
* f5eb5a4 - 2022-10-19 : Added functions and class documentation
*   8a4ea0c - 2022-10-19 : Merge pull request #2 from JazminSantibanez/TableVars
|\
| * 7a690dd - 2022-10-19 : Implemented directory of functions class
* |   e17c2ac - 2022-10-19 : Merge pull request #1 from JazminSantibanez/TableVars
|\ \
| |/
| * cf368e6 - 2022-10-19 : Added table for variables
|/
* 7bffeec - 2022-10-18 : Update README.md
* 005a8ca - 2022-10-17 : Fixed shift/reduce conflicts.
* 54dac79 - 2022-10-17 : Fixed parser, test1C passed.
* 1a130be - 2022-10-12 : Updated proposal doc
* a282157 - 2022-10-12 : Update .gitignore
* 6357322 - 2022-10-12 : Added basic considerations for semantic
* 25d80bf - 2022-10-12 : Fixed spelling errors, updated cuboSem
* 7660376 - 2022-10-11 :  Rearrangement of folders
* 4a74f4a - 2022-10-11 : Added new parsing rules and semantic cube
* 962b776 - 2022-10-10 : Update of lexer and parser
* bf56fb4 - 2022-10-05 : Avance 1
* d4c58b0 - 2022-10-05 : Initial commit
```