MyRLanguage

Tecnológico de Monterrey

A01250658

Juventino Gutierrez Romo

23 de Noviembre 2023

Índice

Descripción del proyecto

El propósito del proyecto es desarrollar e implementar nuestro propio lenguaje de programación, el cual he desarrollado utilizando el lenguaje de programación Python. El proceso comienza al recibir una entrada de texto que consiste en una secuencia de tokens que siguen un orden específico definido por las reglas del lenguaje. Luego, utilizamos la sintaxis diseñada para interpretar estos tokens y generar la estructura del programa.

Una vez que hemos realizado el análisis sintáctico, entramos en la fase de semántica, donde generamos el código intermedio. Este código actúa como una representación de las operaciones realizadas en el programa. Finalmente, nuestra máquina virtual interpreta este código intermedio y ejecuta las operaciones correspondientes.

En resumen, nuestro objetivo es crear un lenguaje de programación propio utilizando Python, donde podemos recibir una entrada de texto con tokens, analizar su sintaxis y generar código intermedio mediante la semántica. A través de nuestra máquina virtual, logramos interpretar y ejecutar las operaciones generadas en el código intermedio

Análisis de requerimientos

- ***Expresiones aritméticas/Lógicas y Relacionales***.
- ***Estatutos de Interacción (Entrada / Salida)***.
- ***Estatutos de Control de Flujo (Ciclos, Decisiones)***.
- ***Elementos de cambio de contexto (Funciones/Métodos parametrizables)***
- ***Manejo de Elementos NO-Atómicos (Vectores)***
- ***Elementos propios del tipo de proyecto (Output gráfico, Estadísticos)***
- ***Pruebas de Cálculo de Factorial y Serie de Fibonacci (en versión cíclica y versión recursiva).***
- ***Pruebas de Sort y un Find para un Vector. (Con estas pruebas validamos trabajo sobre elementos estructurados)***

Aprendizajes

Durante este proyecto hubo demasiados problemas que al final se pudieron solucionar, con trabajo y esfuerzo se pueden resolver. Pero con el compilador aprendí demasiado a cómo debuggear ya que tenía muchos problemas con la generación de cuádruplos, pero de conceptos computacionales aprendí sobre estructura de datos y algoritmos y un mejor entendimiento sobre compiladores. Todavía hay muchas cosas por aprender, ya sea sobre compiladores, estructuras de datos o algoritmos, pero este proyecto me dio una base sólida para seguir explorando y profundizando en estos temas.

Avances:

1 Sep 2023: Planeación del proyecto
22 Sep 2023: Léxico y sintaxis listos
29 Sep 2023: revisión de Lex y Parser
8 Oct 2023: Cubo Semántico Listo
16 Oct 2023: Avance de semántica básica
23 Oct 2023: Avance de semántica aritmética
7 Nov 2023: Avance de Asignación y llamada
14 Nov 2023: Avance de Direcciones de Memoria
20 Nov 2023: Generación de estatutos terminada
26 Nov 2023: Generación de Código de funciones y VM lista

Descripción del Lenguaje

Nombre del Lenguaje: MyRLanguage

Descripción genérica:

Es un lenguaje de programación con el funcionamiento base como c,este permite:

- Crear y utilizar variables tipo int, float y char.
- Puede leer y escribir datos como int, float y char.
- Hace operaciones aritméticas, lógicas, relacionales.
- Cuenta con estatutos de Decisión.
- Declara Funciones retornables tipo int, float, char y no retornables como void.
- Maneja elementos como arreglos

Errores:

-
- Variable already declared
- Function already declared
- Stack overflow
- Variable not declared
- Parentesis mismatch
- Type mismatch
- Function has to return a value
- Function should not have a return
- Parameter types do not match
- Number of parameters does not match
- Dimension must be greater than 0
- variable does not have the size
- number of dimensions mismatch the variable

Descripción del Compilador

Librerías

Para desarrollar el compilador, se utilizó la biblioteca PLY (Python Lex-Yacc), que es una implementación en Python de las herramientas Lex y Yacc. PLY proporciona una manera sencilla y flexible de

construir analizadores léxicos (lexers) y sintácticos (parsers) para el procesamiento de lenguajes de programación.

Analisis Lexico

Token	Pattern
PROGRAM	'Program'
VAR	'Var'
FUNCTION	'Function'
INT	‘int’
FLOAT	‘float’
CHAR	‘char’
BOOL	‘bool’
VOID	‘void’
MAIN	'main'
RETURN	'return'
READ	'Read'
WRITE	'Write'
IF	'If'

THEN	'Then'
ELSE	'Else'
WHILE	'While'
DO	'Do'
FOR	'For'
TO	'To'
SEMICOLON	';'
COMA	','
COLON	':'
L_BRACE	'{'
R_BRACE	'}'
L_PAREN	'('
R_PAREN	')'
L_BRACKET	'['
R_BRACKET	']'
EQUAL	'='
PLUS	'+'

MINUS	'-'
MULT	'*'
DIV	'/'
AND	'&'
OR	'|'
LT	'<'
GT	'>'
LEQ	'<='
GEQ	'>='
EQ	'=='
INT	'\d+'
FLOAT	'\d+\.\d+'
CHAR	'\'.*?\''
STRING	'\".*?\"'
ID	'[a-zA-Z_][a-zA-Z_0-9]*'

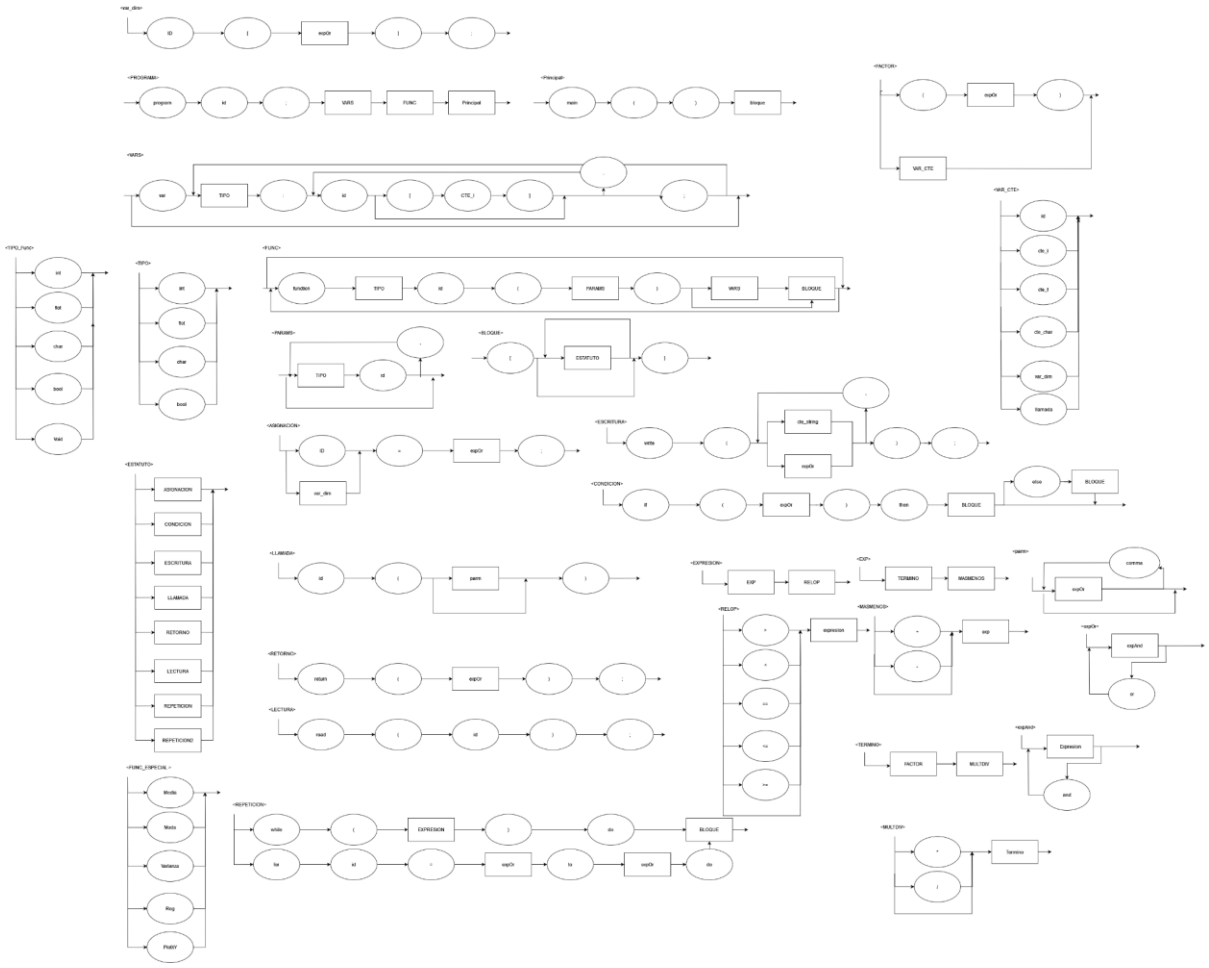
Análisis de Sintaxis

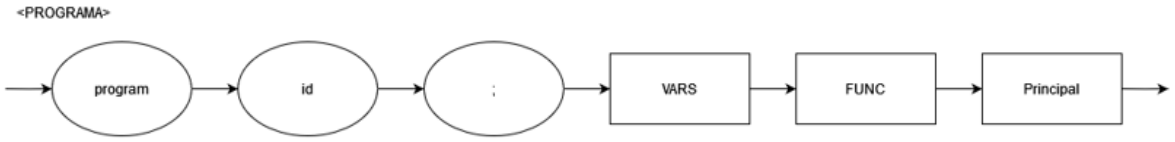
Gramática Formal

1. **Start Rule**
    - PROGRAMA -> 'PROGRAM' 'ID' 'SEMICOLON' vars2 func2 principal
2. **Variable Declarations**
    - vars -> 'VAR' tipo 'COLON' id_list 'SEMICOLON'
    - vars2 -> vars vars2 | empty
3. **Identifier Lists**
    - id_list -> id_list 'COMMA' 'ID' array | 'ID' array
4. **Array Handling**
    - array -> 'L_BRACKET' 'CTE_I' 'R_BRACKET' | empty
5. **Type Specifications**
    - tipo -> 'INT' | 'FLOAT' | 'BOOL' | 'CHAR'
6. **Function Definitions**
    - func -> 'FUNCTION' tipo_func 'ID' 'L_PAREN' params 'R_PAREN' vars2 'L_BRACE' estatuto_rep 'R_BRACE'
    - func2 -> func func2 | empty
    - tipo_func -> 'INT' | 'FLOAT' | 'CHAR' | 'BOOL' | 'VOID'
7. **Function Parameters**
    - params -> tipo 'ID' params2 | empty
    - params2 -> 'COMMA' tipo 'ID' params2 | empty
8. **Main Function**
    - principal -> 'MAIN' start 'L_PAREN' 'R_PAREN' bloque
9. **Block and Statement Repetition**
    - bloque -> 'L_BRACE' estatuto_rep 'R_BRACE'
    - estatuto_rep -> estatuto estatuto_rep | empty
10. **Statements**
    - estatuto -> asignacion | condicion | escritura | llamada | retorno | lectura | repeticion | repeticion2
11. **Assignments and Function Calls**
    - asignacion -> 'ID' 'EQUAL' expOr 'SEMICOLON' | var_dim 'EQUAL' expOr 'SEMICOLON'
    - llamada -> 'ID' 'L_PAREN' parm 'R_PAREN'
12. **Return, Read, and Write Statements**
    - retorno -> 'RETURN' 'L_PAREN' expOr 'R_PAREN' 'SEMICOLON'
    - lectura -> 'READ' 'L_PAREN' 'ID' 'R_PAREN' 'SEMICOLON'
    - escritura -> 'WRITE' 'L_PAREN' escritura_rep 'R_PAREN' 'SEMICOLON'
    - escritura_rep -> escritura_rep 'COMMA' escritura_aux | escritura_aux
    - escritura_aux -> 'CTE_S' | expOr
13. **Conditional and Loop Structures**

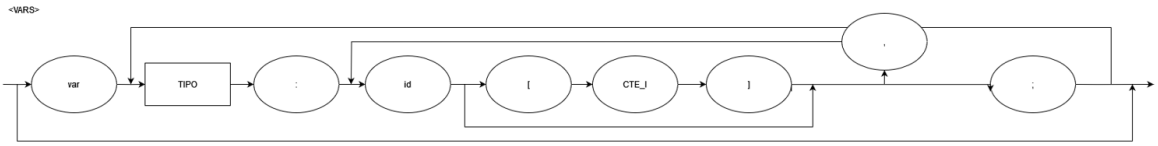
- condicion -> 'IF' 'L_PAREN' expOr 'R_PAREN' 'THEN' bloque else_aux
- else_aux -> 'ELSE' bloque
- repeticion -> 'WHILE' 'L_PAREN' expOr 'R_PAREN' 'DO' bloque
- repeticion2 -> 'FOR' 'ID' 'EQUAL' expOr 'TO' expOr 'DO' bloque

14. **Parameter Handling in Function Calls**
    - parm -> expOr parm2 | empty
    - parm2 -> 'COMMA' expOr parm2 | empty
15. **Expressions**
    - expOr -> expAnd 'OR' expOr | expAnd
    - expAnd -> expresion 'AND' expAnd | expresion
    - expresion -> exp relop
    - relop -> 'GT' expresion | 'LT' expresion | 'EQ' expresion | 'LEQ' expresion | 'GEQ' expresion | empty
    - exp -> termino masmenos
    - `masmenos -> 'PLUS' exp





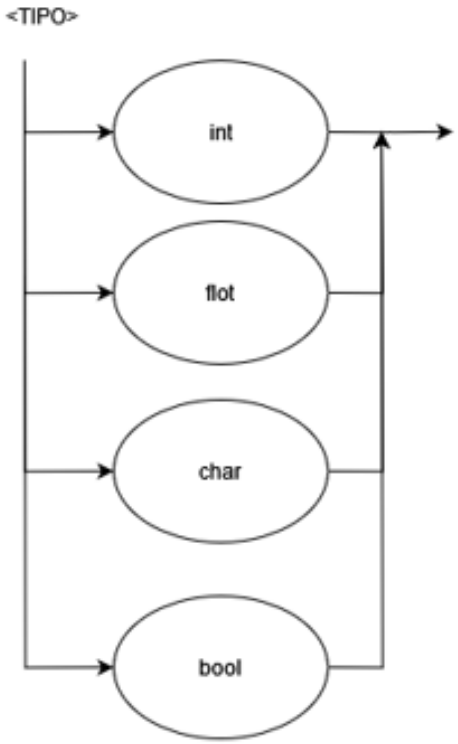
1-create_dirfunc: Crea una nueva función en el directorio de funciones



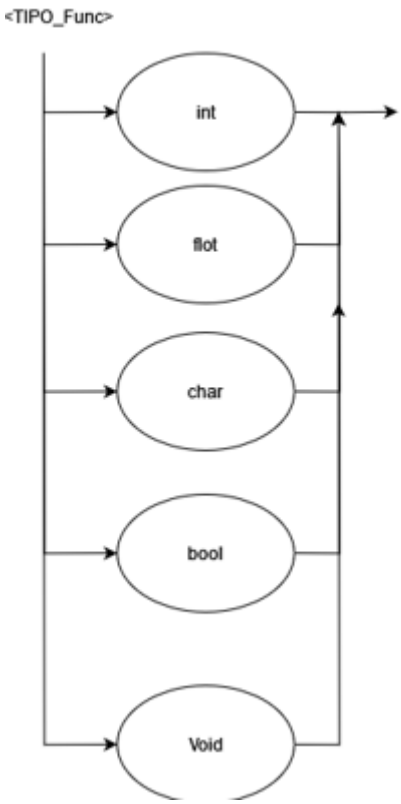
2-addVar:Esta función agrega una variable al diccionario de variables de la función actual

3- addDim:Añade una dimensión a una variable en la pila de operandos

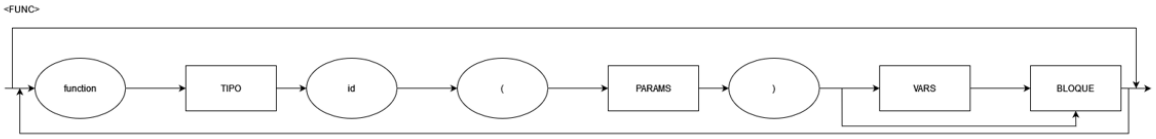
4-jumpAddr:Salto a dirección de salto



5-current_type:Establece la variable global CurrentType con el valor de p[-1]



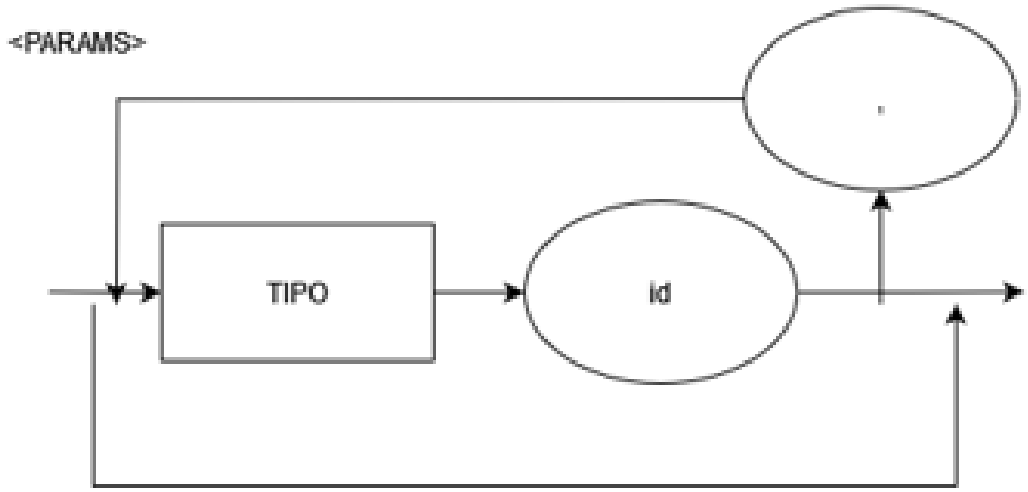
6-current_type:Establece la variable global CurrentType con el valor de p[-1]



7-addfunc:Añade una función al diccionario de funciones

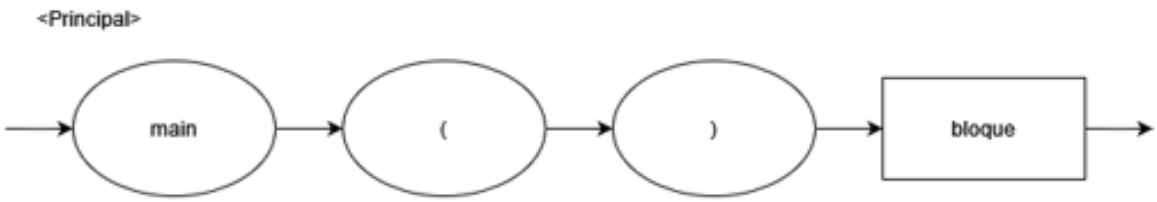
8-funcJump:Establece la dirección de inicio de una función en el diccionario dirFunc,actualiza el tamaño de la función

9-endFunc:Agrega el cuádruplo 'ENDFUNC' a la lista de cuádruplos



10-addvar:

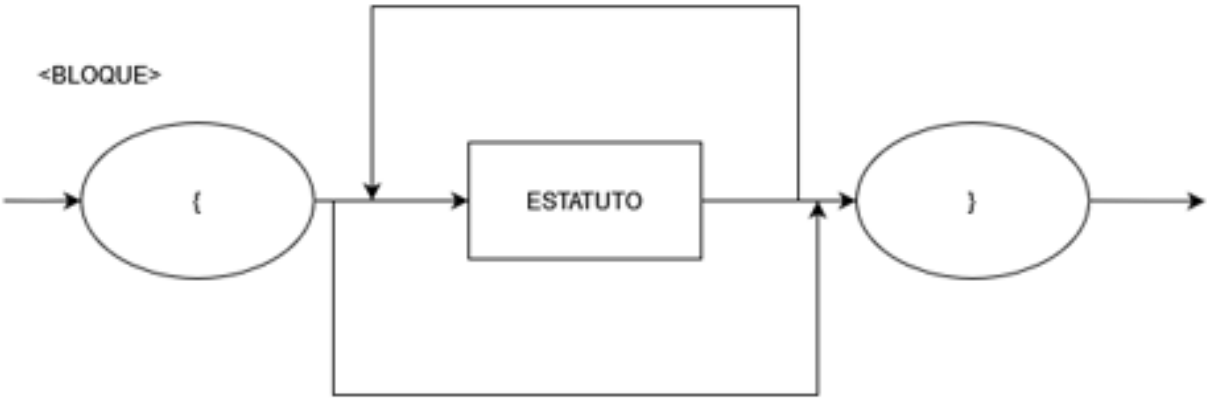
11-updateParams:

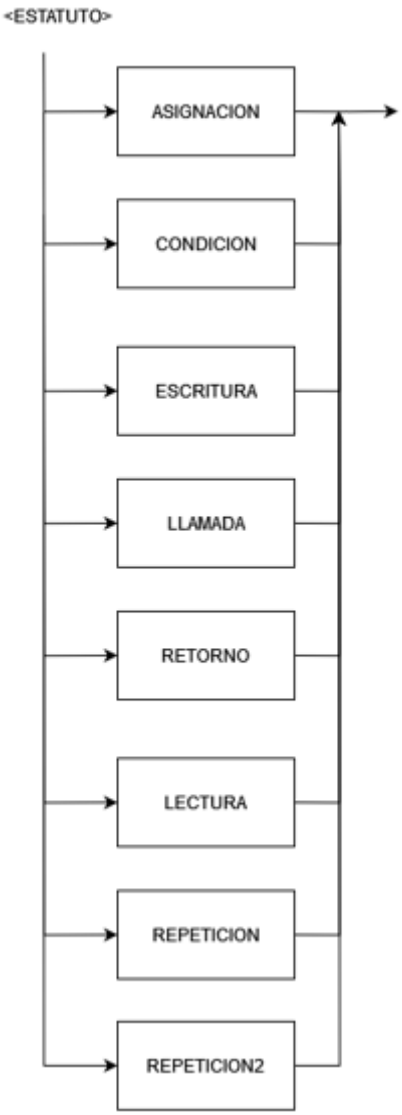


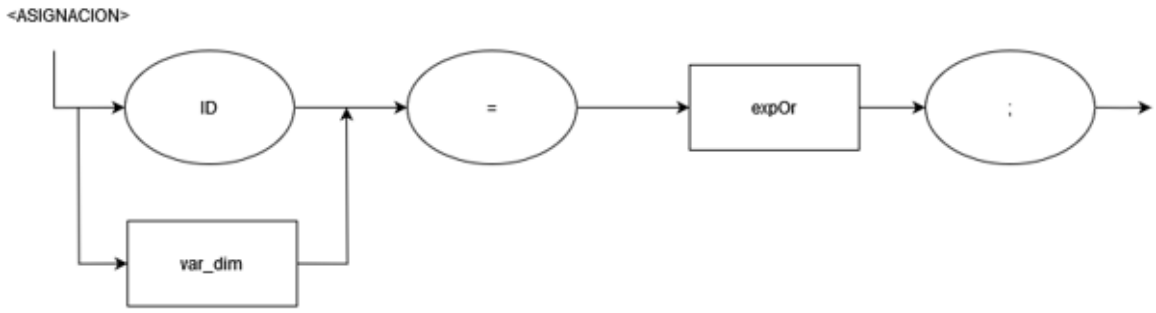
12-start:Esta función agrega una variable al diccionario de variables de la función actual

13-funcChange:Cambia la función actual a "global"

14-endProc: agrega el cuádruplo ENDPROC a la lista de cuádruplos


<BLOQUE>
{
ESTATUTO
}

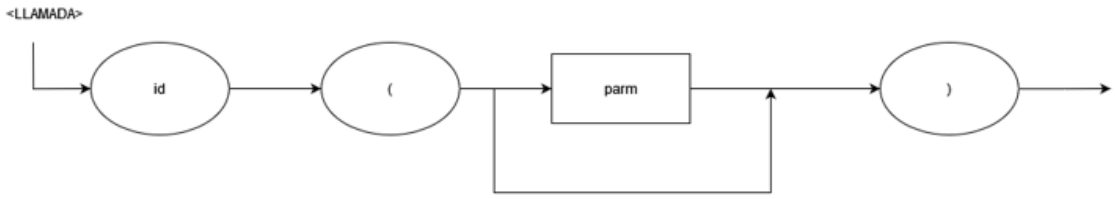

<ESTATUTO>
ASIGNACION
CONDICION
ESCRITURA
LLAMADA
RETORNO
LECTURA
REPETICION
REPETICION2



15-stack_operand_id:maneja la operación de apilar un operando identificador en la pila de operandos

16-stack_operator:Añade el operador a la pila de operadores

17-np_asignacion: agrega el cuádruplo de asignacion a la lista de cuádruplos



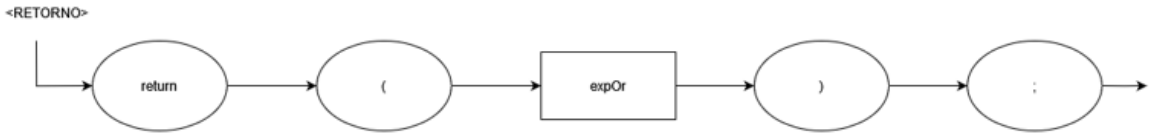
18-llamadaEra: agrega el cuádruplo ERA a la lista de cuádruplos

19-fakebottom: agrega un paréntesis abierto "(" a la pila de operadores

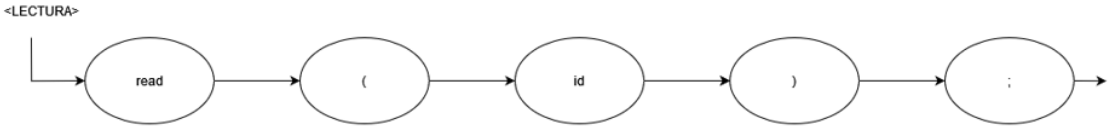
20-checkParamNum:Verifica si el número de parámetros coincide

21-checkparentesis:Verifica si hay un paréntesis en la pila de operadores. Si hay un paréntesis de apertura, lo elimina de la pila.

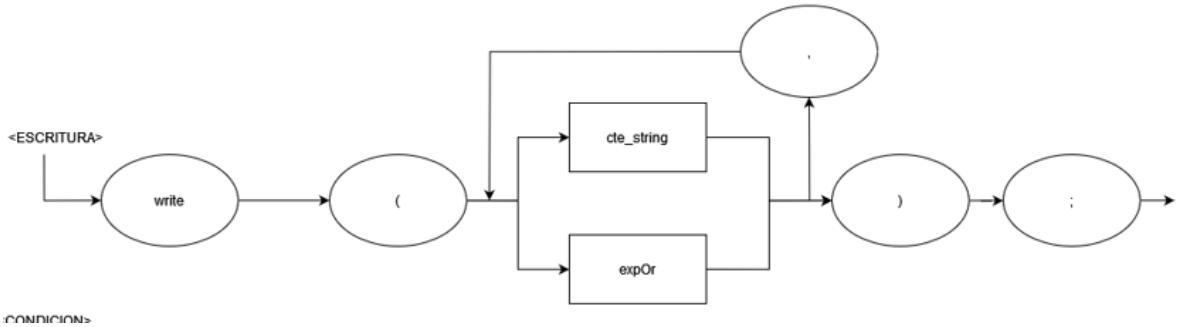
22-Gosub:



23-np_return: agrega el cuádruplo Return a la lista de cuádruplos

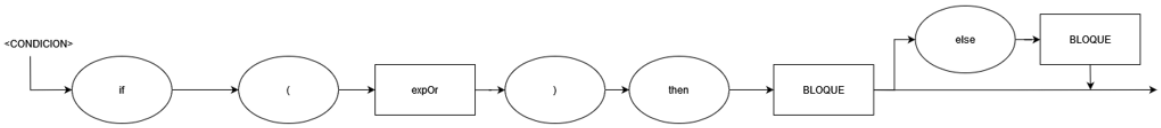


24-np_read:agrega el cuádruplo READ a la lista de cuádruplos



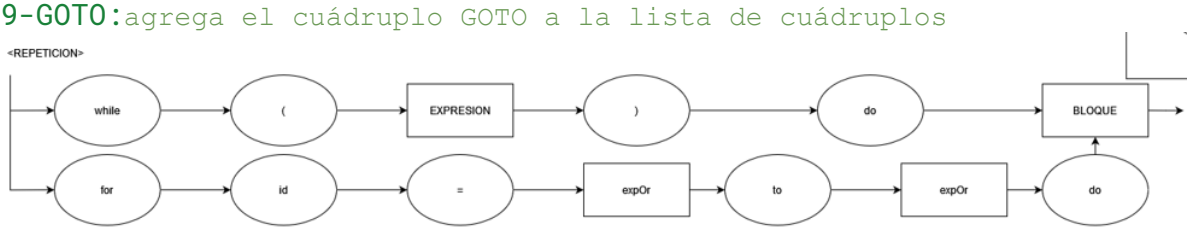
25-np_print: agrega el cuádruplo PRINT a la lista de cuádruplos

26-printString:Imprime una String, agrega el cuádruplo PRINT a la lista de cuádruplos



28-GOTOF:agrega el cuádruplo GOTOF a la lista de cuádruplos

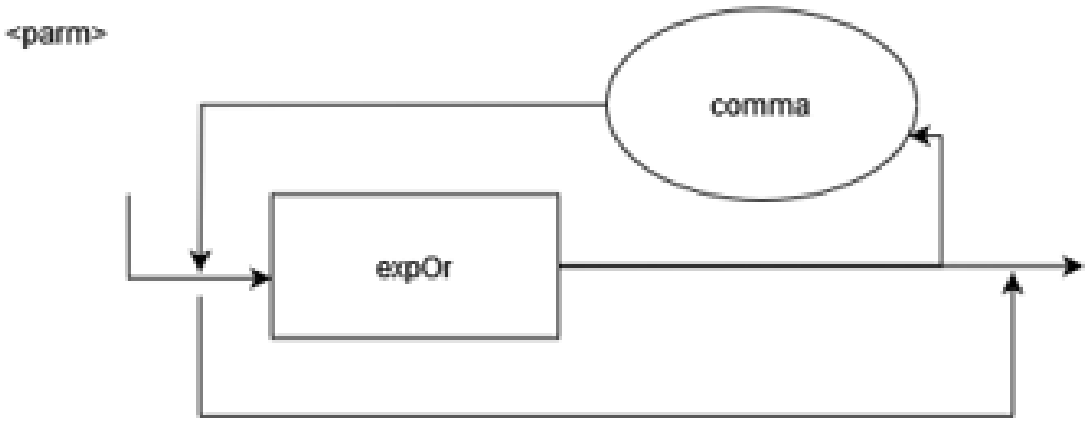
29-GOTO:agrega el cuádruplo GOTO a la lista de cuádruplos



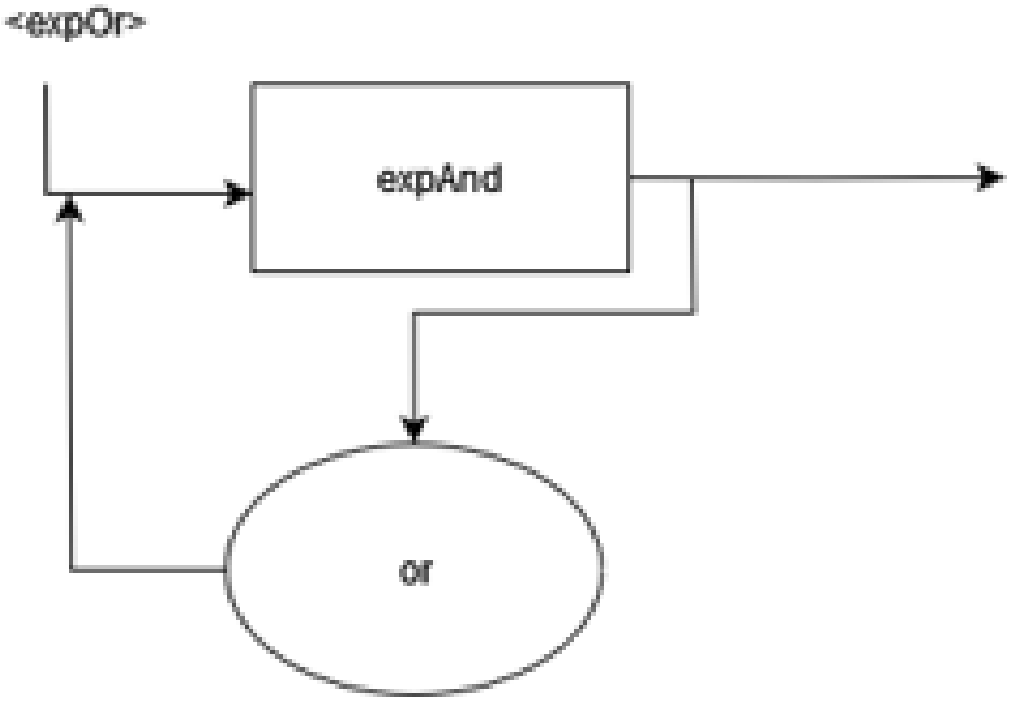
30-addJump:Añade un salto a la pila de saltos

31-Gotof:agrega el cuádruplo GOTOF a la lista de cuádruplos

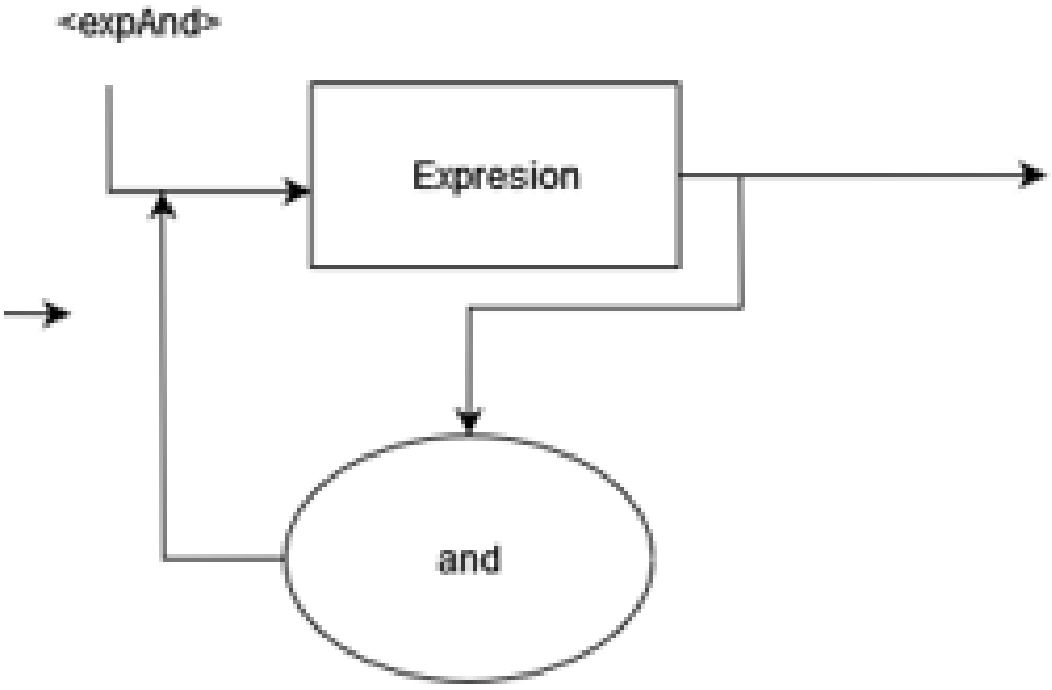
32-endWhile:Desapila los elementos necesarios de las pilas y genera el cuádruplo GOTO para finalizar un bucle while



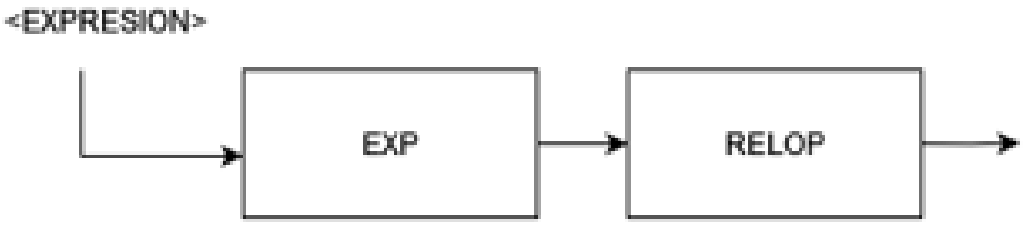
33-CheckParam:comprobar si el tipo del parámetro pasado coincide con el tipo esperado, si si, agrega el cuádruplo PARAM a la lista de cuádruplos



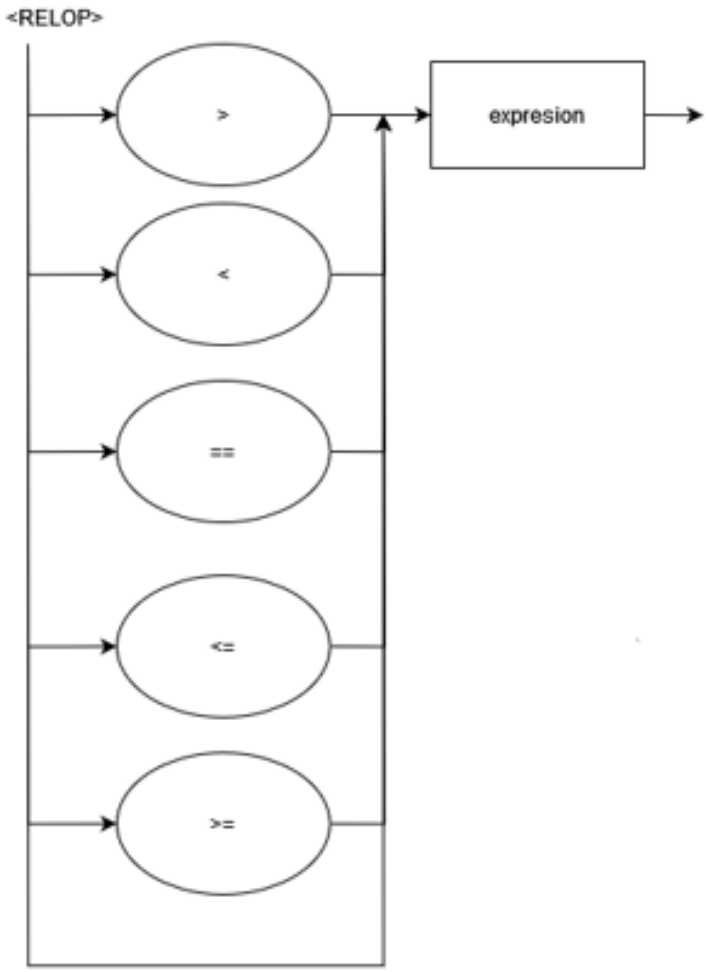
34-checkAndOr:agrega los cuádruplos AND OR a la lista de cuádruplos
35- stack_operator:Añade el operador a la pila de operadores



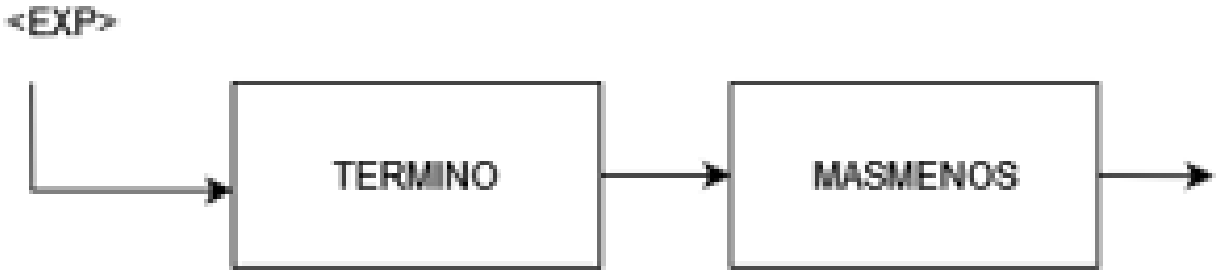
36-checkAndOr:agrega los cuádruplos AND OR a la lista de cuádruplos
37- stack_operator:Añade el operador a la pila de operadores



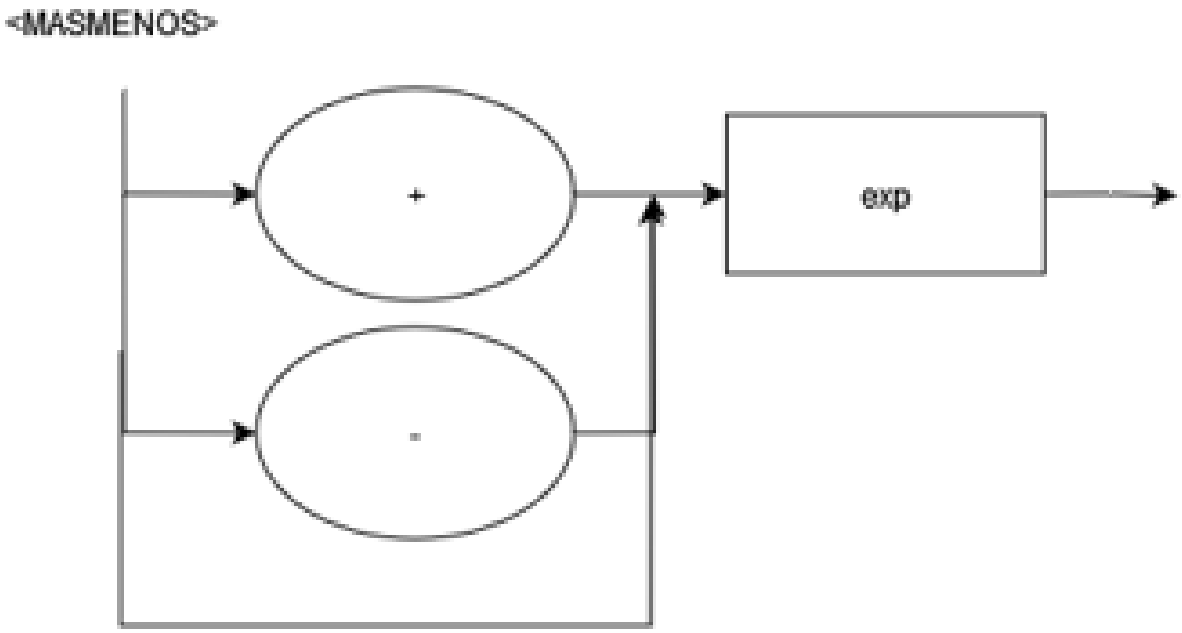
38-checkrelop:agrega los cuádruplos comparadores a la lista de cuádruplos



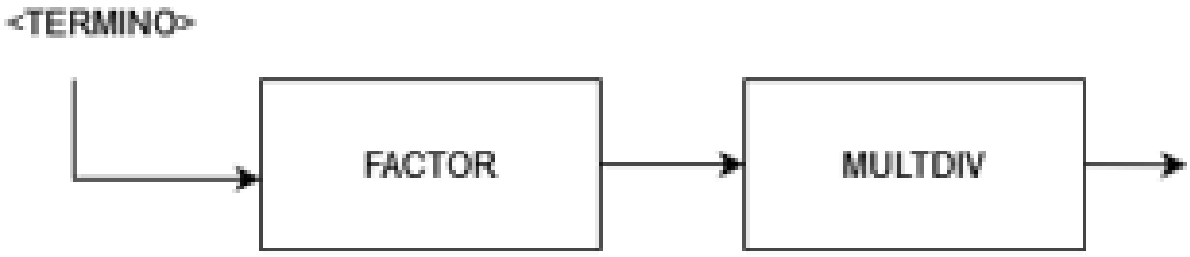
39-stack_operator:Añade el operador a la pila de operadores



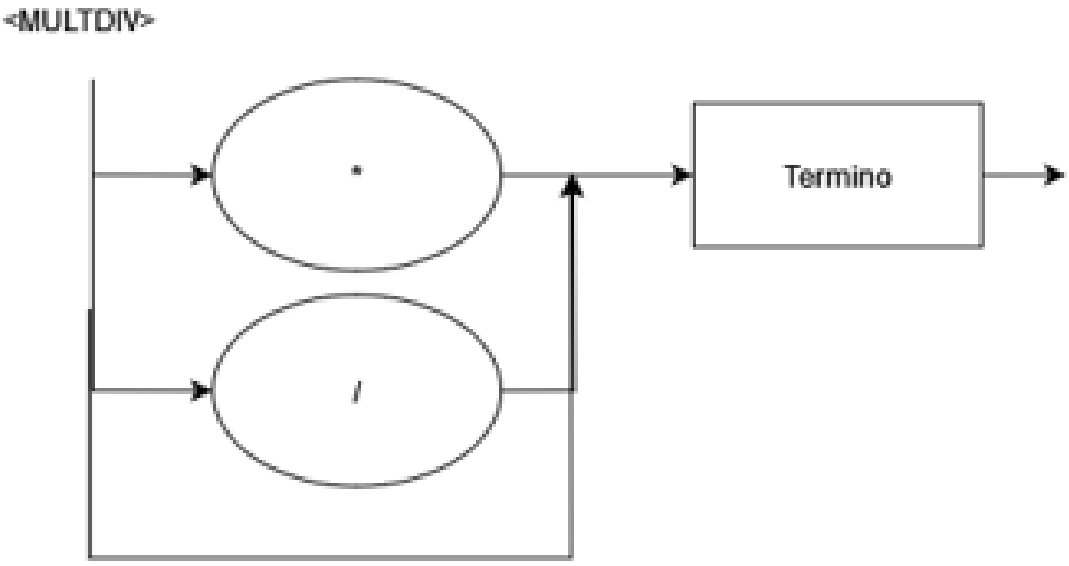
40-checkexp:agrega los cuádruplos + - a la lista de cuádruplos



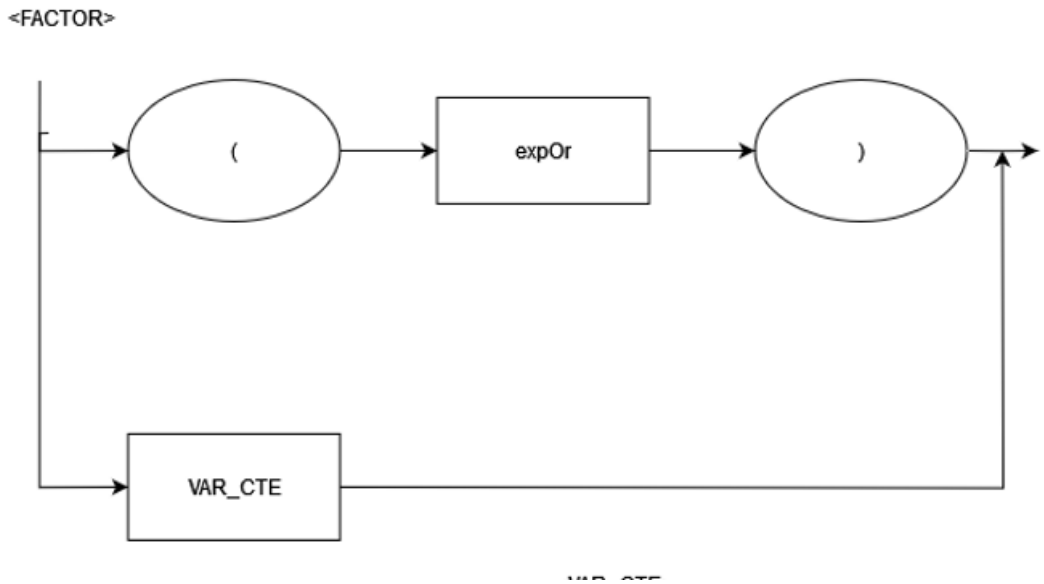
41-stack_operator:Añade el operador a la pila de operadores



42-checkterm: agrega los cuádruplos * / a la lista de cuádruplos

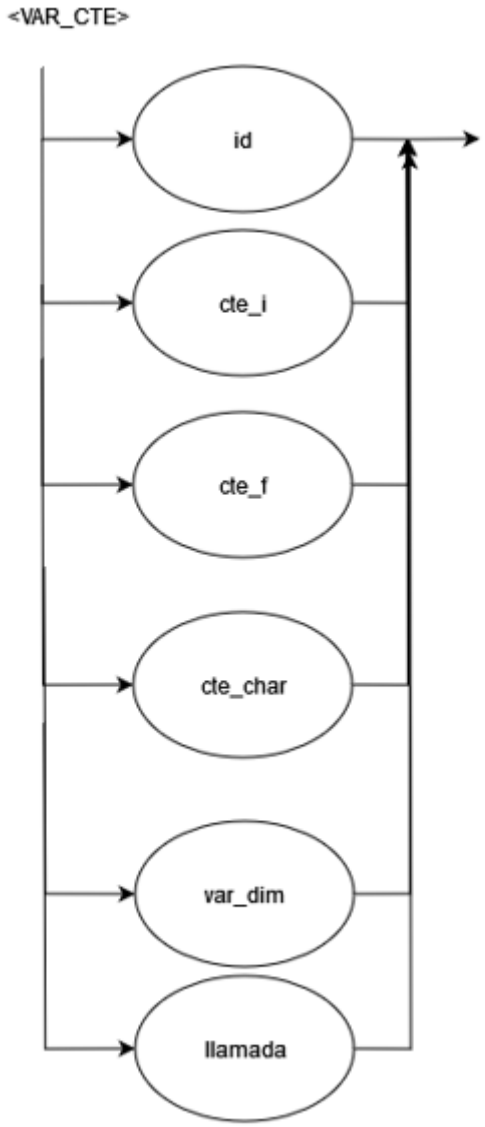


43-stack_operator:Añade el operador a la pila de operadores



44-fakebottom: agrega un paréntesis abierto "(" a la pila de operadores

45-checkparentesis: Verifica si hay un paréntesis en la pila de operadores. Si hay un paréntesis de apertura, lo elimina de la pila.

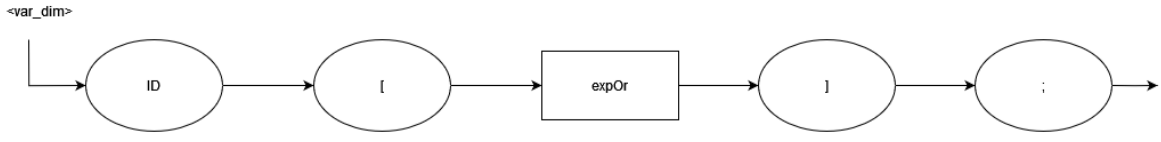


46-stack_operand_id: maneja la operación de apilar un operando identificador en la pila de operandos

47-stack_operand_int: Apila un operando entero en la pila de operandos

48-stack_operand_float: Apila un operando flotante en la pila de operandos

49-stack_operand_char: Apila un operando carácter en la pila de operandos



50-stack_operand_id: maneja la operación de apilar un operando identificador en la pila de operandos

51-verDim: Verifica si la variable tiene la dimensión adecuada
52-fakebottom:agrega un paréntesis abierto "(" a la pila de operadores

53-cuadVer: Verifica si el valor en la cima de la pila de operandos está dentro de los límites establecidos
54-checkparentesis:Verifica si hay un paréntesis en la pila de operadores. Si hay un paréntesis de apertura, lo elimina de la pila.

55-verDimNum: Verifica si el número de dimensiones coincide con la variable
56-cuadVarDim:Función que maneja la generación de cuádruplos para variables dimensionadas

Cubo Semantico

Type 1	Type 2	Operator	Resulting Type
Int	Int	+, -, *, /	Int
Int	Float	+, -, *, /	Float
Int	Float	==, !=	Int
Int	Char	+, -, *, /	Error
Float	Int	+, -, *, /	Float
Float	Float	+, -, *, /	Float
Float	Float	==, !=	Int
Float	Char	-	Error
Char	Int	-	Error

Char	Float	-	Error
Char	Char	==, !=	Int
Char	Char	+, -, *, /	Error

1. **Tabla de variables**: Esta es una estructura de datos que contiene información sobre todas las variables utilizadas en el programa. Para cada variable, se registra su nombre, su tipo de dato y su dirección de memoria. En este código, la tabla de variables es un diccionario que almacena las variables de la función junto con su tipo y su dirección de memoria.
2. **Tabla de constantes**: Esta es similar a la tabla de variables pero para constantes. La tabla de constantes es un diccionario que contiene constantes de diferentes tipos (Entero, Flotante, Carácter, Booleano) junto con sus direcciones de memoria.
3. **Cuádruplos**: Los cuádruplos son una forma común de representar las instrucciones en la generación de código intermedio en un compilador. Cada cuádruplo contiene cuatro campos: operador, operando 1, operando 2 y resultado. Los cuádruplos representan una operación que se realizará. En este compilador, el array cuádruplo se llena con objetos de tipo cuádruplo que representan las instrucciones del programa.
4. **Pilas**: Las pilas se utilizan para varias funciones en este compilador, como mantener un seguimiento de las variables de ciclo , operadores , términos y tipos . Las pilas son útiles en la compilación porque permiten un acceso rápido y ordenado a la información, y se utilizan comúnmente en la resolución de expresiones y en la implementación de estructuras de control, como ciclos y condicionales.

En cuanto a la **arquitectura de la memoria**, este código tiene una división de la memoria en distintas secciones para diferentes tipos de datos y usos:

- Memoria global (para enteros, flotantes, booleanos y caracteres).
- Memoria local (para enteros, flotantes, booleanos y caracteres).
- Memoria constante (para enteros, flotantes, booleanos y caracteres).
- Memoria temporal para enteros, flotantes, booleanos y caracteres.
- Memoria para Strings.

Estos valores de memoria representan el comienzo de cada sección de memoria para cada tipo de dato. Durante la ejecución del programa, estas direcciones de memoria se utilizan para almacenar y recuperar valores.

Direcciones de memoria:

```
global_int = 1000
global_float = 2000
global_char = 3000
global_bool = 4000
local_int = 5000
local_float = 6000
local_char = 7000
local_bool = 8000
const_int = 9000
const_float = 10000
const_char = 11000
const_bool = 12000
temp_int = 13000
temp_float = 14000
temp_char = 15000
temp_bool = 16000
stringMemory = 17000
pointerInt = 18000
pointerFloat = 19000
```

Descripción de la Máquina Virtual

Lenguaje y Librerías usadas:

La máquina virtual se desarrolló en windows y fue escrita en el lenguaje python.

La máquina virtual implementada tiene como objetivo ejecutar los cuádruplos generados durante la compilación del programa.

Administración de Memoria

1. UpdateMemory: La función updateMemory está diseñada para actualizar diferentes tipos de memoria basándose en la dirección de memoria (dir) y el valor (val) proporcionados. La función utiliza una serie de declaraciones if y elif para determinar qué tipo de memoria actualizar en función del rango en el que cae la dirección de memoria. Los tipos de memoria están representados por diferentes diccionarios: intMemory, floatMemory, charMemory, boolMemory y pointVals.

2. Get_val:La función get_val(dir, currentLevel) en Python está diseñada para obtener el valor almacenado en una dirección de memoria específica (dir) en un nivel de memoria dado (currentLevel).La función utiliza una serie de declaraciones if y elif para determinar qué tipo de memoria consultar en función del rango en el que cae la dirección de memoria. Los tipos de memoria están representados por diferentes diccionarios: intMemory, floatMemory, charMemory, boolMemory y Constants.

1000-1999 Int	2000-2999 Float	3000-3999 Char	4000-4999 BOOL	5000-5999 Int	6000-6999 Float	7000-7999 Char	8000-8999 BOOL	9000-12999 Constan	10000-19999 String

								ts	Global

13000-13999 Int	14000-14999 Float	15000-15999 Char	16000-16999 BOOL	17000-17999 Constants	18000-18999 Ponters	19000-19999 Pointers			

Pruebas

Manual de Usuario

1. Instala Python
2. Descargar el repositorio
3. Abre el folder
4. Abre la ubicación del folder ‘compilador’.
5. Correr la main

Video:

https://youtu.be/PneG8Os0mdI

Github:

https://github.com/juvengtz/Compilador