Documentación de Bernardo: Lenguaje de Programación

Desarrollador:

Bernardo Montemayor Hernández
A01194086

22 de Noviembre del 2022

Diseño de Compiladores

Índice

1.1- Propósito y Alcance

Este proyecto tiene como propósito ejecutar y compilar un lenguaje de programación desarrollado en base a R. El lenguaje propuesto si es similar a R, pero su capacidad, alcance y rango de acciones está limitado a comparación del lenguaje original, puesto que este lenguaje es simplemente un proyecto desarrollado individualmente a nivel avanzado por un estudiante en su último semestre de la universidad. El lenguaje tiene mucho espacio para ser optimizado.

Este lenguaje tiene soporte hacia la realización de operaciones de tipo suma, resta, multiplicación, división, manejo de estatutos condicionales, manejo de estatutos cíclicos, llamado de funciones y manejo de arreglos de una dimensión.

El proyecto contiene una cantidad considerable de manejo de errores pero no se llenó a su plenitud con todas las consideraciones, por lo que es probable que al probar ciertos inputs se podrán encontrar fallos de tipo o de gramática simple para asignación de ciertos elementos.

1.2- Análisis de Requerimientos

El programa debe ser capaz de compilar y ejecutar:

- Expresiones aritméticas, lógicas y relacionales
- Estatutos de interacción (entrada / salida)
- Estatutos de control de flujo (ciclos y decisiones)
- Elementos de cambio de contexto (funciones parametrizables)
- Manejo de elementos no tóxicos (arreglos)
- Funciones estadisticas y graficas

El programa ejecuta de forma eficiente con manejo, asignación y ejecución de memoria, sin desperdiciar recursos con estructuras de datos redundantes.

1.3- Descripción del Proceso

Bitácora

Avance#	Fecha	Descripción
1	3/Oct/2022	El avance contiene tokens, palabras reservadas y gramática de expresiones regulares.
2	12/Oct/2022	No hubo mucho avance en esta semana pero se agregó lo que se creía que era cierta gramática correcta y el cubo semántico.
3	17/Oct/2022	No hubo avance en esta entrega.
4	4/Nov/2022	Esta entrega se corrigió la gramática, se desarrollo el objeto de cuadruplos correctamente, se corrigió el desarrollo del cubo semántico, se generaron los estatutos cíclicos y condicionales, se asignaron reglas semánticas relacionadas con el objeto de cuádruplos y genera los cuádruplos en base a la gramática.
5	7/Nov/2022	En esta entrega se generó el código intermedio de funciones para el manejo de errores y sus creaciones de cuádruplos.
6	15/Nov/2022	Se le agregó memoria virtual, asignación de espacio de memoria para las variables y procesos relacionados a su scope y su tipo de variable.
-	18/Nov/2022	No hubo entrega pero para esta fecha se desarrolló la máquina virtual para el análisis de cuadruplos y ejecución en su base, manejo de memoria virtual con sus variables dependiendo del scope, tomando en cuenta funciones. También se comenzaron a desarrollar las listas.
7	22/Nov/2022	Entrega final, se desarrolló el manejo de memoria en la máquina virtual para recursión con pase de parámetros y memoria en cada scope, se terminó el desarrollo de las listas con una simulación de apuntadores y se agregaron las funciones estadísticas para los elementos propios del proyecto. También se desarrollaron las pruebas y la documentación.

No hubo repositorio en GitHub, por lo que no se pueden proporcionar los commits.

Reflexión Individual

El proyecto está desarrollado de múltiples estructuras relacionadas y yo creo que lo más importante que aprendí es el poder reconocer la ubicación de cada elemento que se debía agregar o editar. Durante el proceso fui siguiendo las instrucciones de las presentaciones y notas proporcionadas por la maestra pero sin contexto no es posible solamente agregar las operaciones que los recursos recomendaban, se necesitaba una comprensión de cómo se van a relacionar los datos en cada estructura y agregar código de intermedio para poder lograr el desarrollo.

Otra cosa importante es que para un proyecto así de grande se me facilito mucho más con el apoyo de la documentación en medio del código con comentarios, es más fácil identificar procesos y relaciones si tienes notas de que y donde se hacen los procesos. Hubo muchos momentos donde me quedaba atorado por horas con un bug y con el apoyo de los comentarios y haciendo una guía de paso por paso siguiendo la ejecución del programa se lograba identificar dónde exactamente era el problema.

Definitivamente fue un problema que el proyecto lo comencé muy tarde, se puede notar con el resultado del compilador al ver ciertos problemas mínimos que se presentan, pero teniendo en cuenta el tiempo que me quedaba para terminarlo decidí enfocarme en poder demostrar que tengo el conocimiento y las habilidades para cumplir con todos los requisitos de la entrega del proyecto en vez de crear un lenguaje óptimo a la perfección, es por eso que los problemas pequeños que tiene mi lenguaje tienen soluciones fáciles que dentro del programa se puede notar que soy capaz de arreglarlos. A fin de cuentas el programa cumple y cualquier “problema” que tiene son para la optimización de la escritura en el lenguaje.

Definitivamente me hubiera gustado empezar desde antes para que hubiera estado hecho con más pasión y poder haberle metido más creatividad a los elementos propios.

2.1- Descripción del Lenguaje

Nombre del lenguaje: Bernardo

El lenguaje soporta el manejo de variables, listas de una dimensión, estatutos condicionales if, if else y estatutos cíclicos while, do while.
También soporta el uso de funciones de tipo entero, flotantes y void con la aceptación de parámetros de tipo variables, junto con retornos.
De la misma manera acepta operaciones de tipo suma, resta, multiplicación, división, mayor que, menor que, igual a, y y o.

Posibles errores

Errores comunes que se pueden presentar son:

- En ciertos casos, mandar a llamar una función dentro de otra función (que no es la misma) puede asignar un valor equivocado a una variable de la función a llamar, dependiendo de si la dirección virtual de la variable de la función anterior es la misma que la de la variable actual.
- Las divisiones principalmente regresan un entero (al hacer el sort siempre regresaba un flotante y no dejaba acceder a la casilla de la lista, se hizo este hotfix para cumplir con esa prueba).
- No se puede usar una función como parámetro para otra función, se debe de almacenar el resultado de la función en una variable auxiliar y utilizar esa auxiliar como parámetro.
- Hay ciertas verificaciones que no se hacen, algunos estatutos no checan si la casilla del arreglo está fuera de su tamaño.

3.1- Equipo de Computo, Lenguaje, Utilerías, Tokens y expresiones regulares

El compilador fue desarrollado en Python, la gramática fue escrita utilizando la herramienta de PLY, la cual también se encargó de manejar el análisis léxico y sintáctico que hacen posible la fase de compilación.

Los tokens aceptados por el lenguaje son:

- ID, PLUS : + , MINUS : ‘-’ , TIMES : ‘*’ , DIVIDE : ‘/’, EQUALS : ‘=’ , SEMICOLON : ‘;’ , INT, FLOAT, STRING, LT : ‘<’ , GT ‘>’ , LTE : ‘<=’ , GTE : ‘>=’ , DOUBLEEQUAL : ‘==’ , NEQUAL : ‘!=’ , AND : ‘&&’ , OR : ‘||’ , LPAREN : ‘(’ , RPAREN : ‘)’ , LBRACE : ‘[’ , RBRACE : ‘]’ , BLOCKSTART : ‘{’ , BLOCKEND : ‘}’ , COLON : ‘:’ , COMMA : ‘,’ y COMMENT : ‘%%’.

Palabras reservadas:

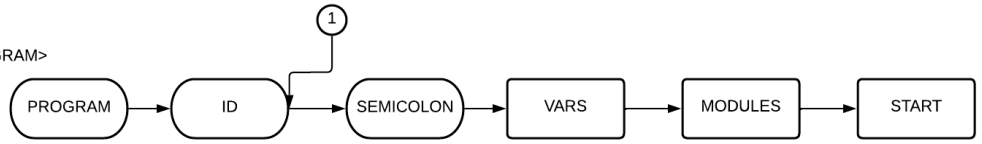
```
'if' : 'IF_K',
'else' : 'ELSE_K',
'while' : 'WHILE_K',
'do' : 'DO_K',
'for' : 'FOR_K',
'return' : 'RETURN_K',
'write' : 'WRITE_K',
'to' : 'TO_K',
'function' : 'FUNCTION_K',
'void' : 'VOID_K',
'vars' : 'VARS_K',
'program' : 'PROGRAM_K',
'main' : 'MAIN_K',
'read' : 'READ_K',
'int' : 'INT_K',
'float' : 'FLOAT_K',
'string' : 'STRING_K',
'mean'  : 'MEAN_K',
'median': 'MEDIAN_K',
'mode'  : 'MODE_K'
'array' : 'ARRAY_K',
'graph' : 'GRAPH_K',
```

Expresiones regulares:

```
t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES = r'\*'
t_DIVIDE  = r'/'
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_LBRACE = r'\['
t_RBRACE = r'\]'
t_BLOCKSTART = r'\{'
t_BLOCKEND = r'\}'
t_EQUALS = r'\='
t_SEMICOLON = r'\;'
t_GT = r'\>'
t_LT = r'\<'
t_LTE = r'\<\='
t_GTE = r'\>\='
t_DOUBLEEQUAL = r'\=\='
t_NEQUAL = r'\!\='
t_AND = r'\&\&'
t_OR = r'\|\|'
t_COMMENT = r'\%\%.*'
t_ignore  = ' \t'
t_COLON = r'\:'
t_COMMA = R'\,'
```

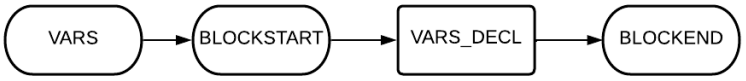
3.2- Gramática, Código intermedio y análisis semántico

<PROGRAM>

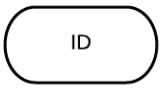


1. Guarda el nombre del programa y lo añade al directorio de funciones

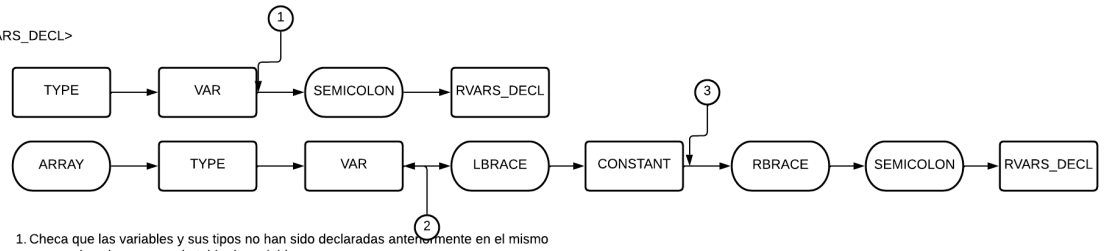
<VARS>



<VAR>



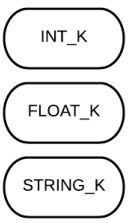
<VARS_DECL>



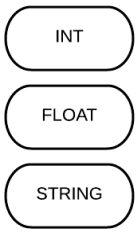
1. Checa que las variables y sus tipos no han sido declaradas anteriormente en el mismo scope y las almacena en la tabla de variables
2. Checar si el array ya existe y agregarlo a la tabla de variables
3. Asignar el tamano del array y sus direcciones virtuales

<RVARS_DECL>

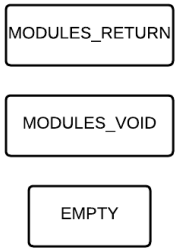
<TYPE>



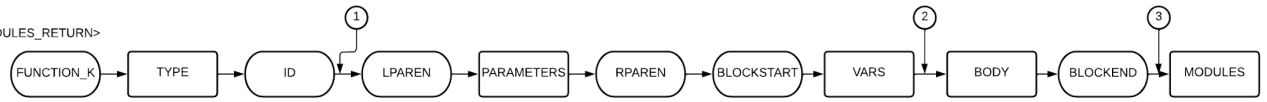
<CONSTANT>



<MODULES>

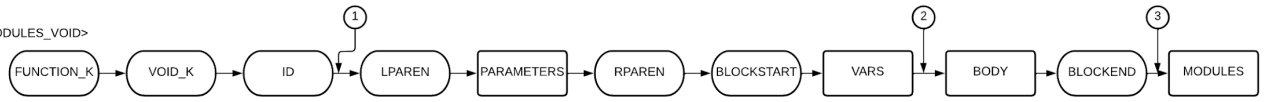


<MODULES_RETURN>



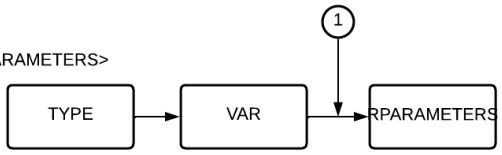
1. Revisa si el nombre de la funcion existe y la mete a la direccion de funciones
2. Mete el tamano y la posicion de quad en el funcdir
3. Crea el cuadruplo de endfunc y resetea la memoria

<MODULES_VOID>



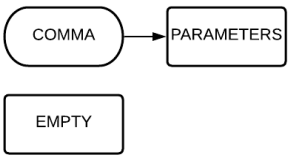
1. Revisa si el nombre de la funcion existe y la mete a la direccion de funciones
2. Mete el tamano y la posicion de quad en el funcdir
3. Crea el cuadruplo de endfunc y resetea la memoria

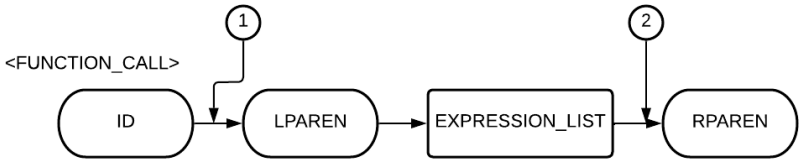
<PARAMETERS>



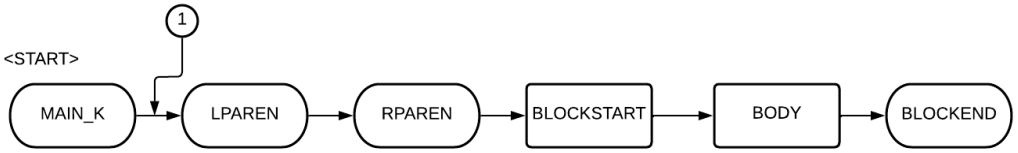
1. Checa que no se hayan declarado las variables y las mete a la tabla de variables

<RPARAMETERS>

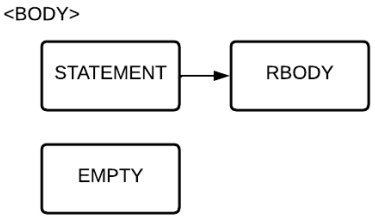


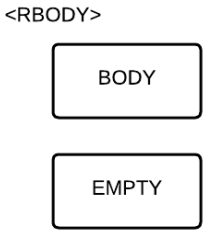


1. Verifica que exista la funcion a llamar y crea el cuadruplo era
2. Verifica que el numero de parametros sea el correcto y crea el cuadruplo gosub

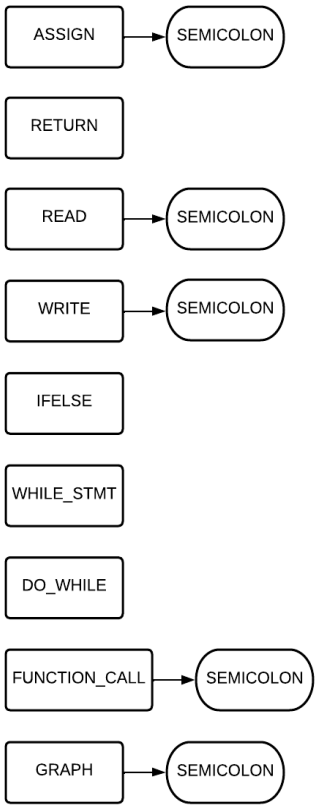


1. Hace pop al jumpstack para el main

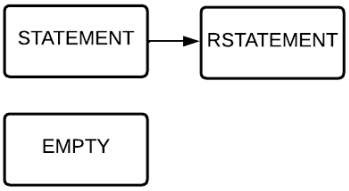




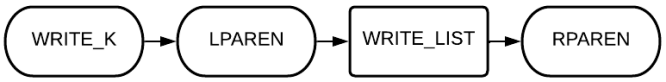
<STATEMENT>



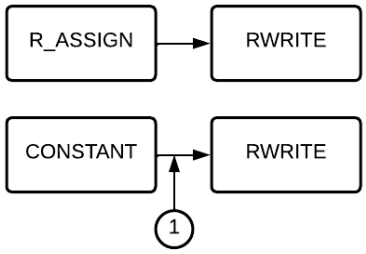
<RSTATEMENT>



<WRITE>

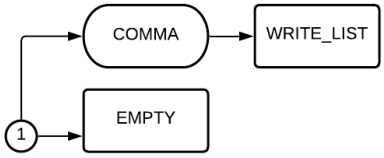


<WRITE_LIST>



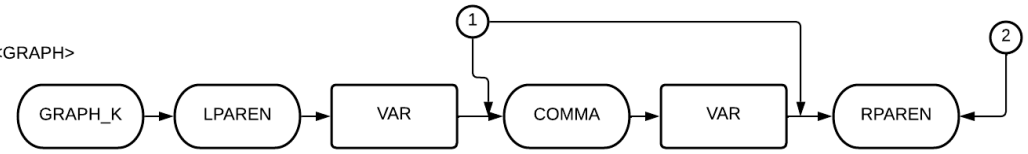
1. Revisa que tipo es la constante, lo mete al stack de tipos y mete el operando al stack de operandos y a si no esta, en la tabla de constantes

<RWRITE>



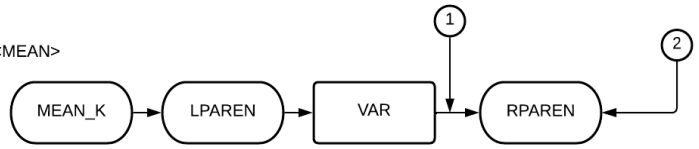
1. Creacion del cuadruplo de write

<GRAPH>

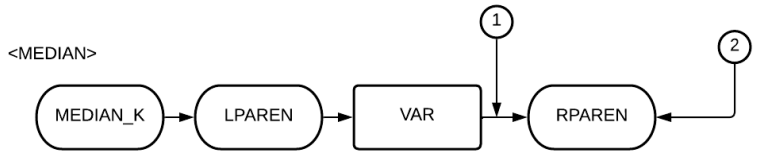


1. Revisa si el parametro es una lista y mete su direccion al stack de operandos
2. Crea el cuadruplo para graficar

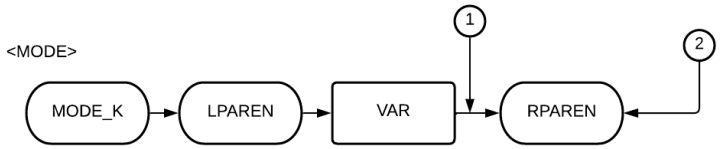
<MEAN>



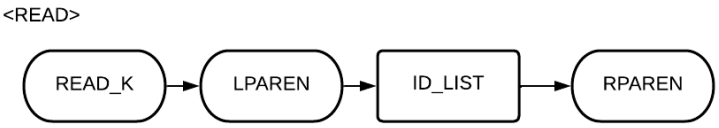
1. Revisa si el parametro es una lista y mete su direccion al stack de operandos
2. Crea el cuadruplo para sacar el mean

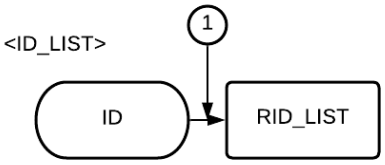


1. Revisa si el parametro es una lista y mete su direccion al stack de operandos
2. Crea el cuadruplo para sacar el median

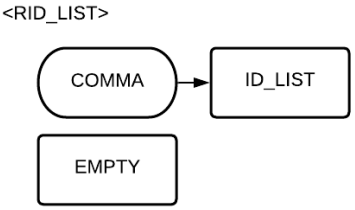


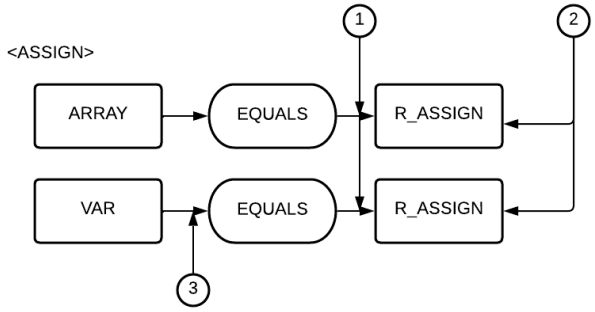
1. Revisa si el parametro es una lista y mete su direccion al stack de operandos
2. Crea el cuadruplo para sacar el mode





1. Genera el cuadruplo de read





1. Mete el equals al stack de poper
2. Revisa el cubo semantico y genera el cuadruplo de asignacion
3. Busca que la variable exista y mete su tipo y operando a sus stacks

<RASSIGN>

EXPRESSION

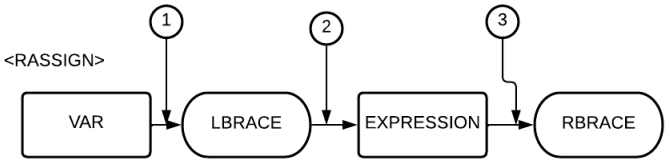
FUNCTION_CALL

ARRAY

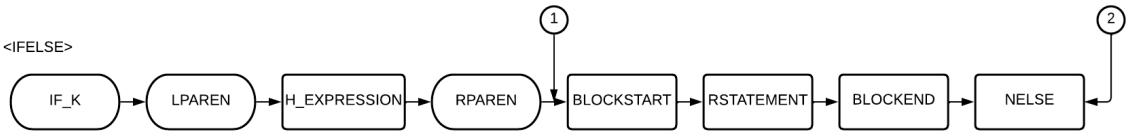
MEAN

MEDIAN

MODE



1. Busca que la variable exista y mete su tipo y operando a sus stacks
2. Guarda el operando del array en una variable
3. Crea el apuntador para navegar a la casilla deseada, creando el cuadruplo GET

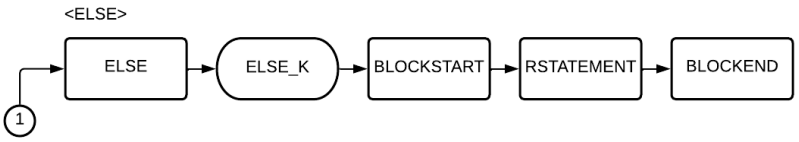


1. Revisa el tipo de la condicion del if, genera el quad y mete el quadcounter a la jumpstack
2. Mete el quadcounter al ultimo brinco

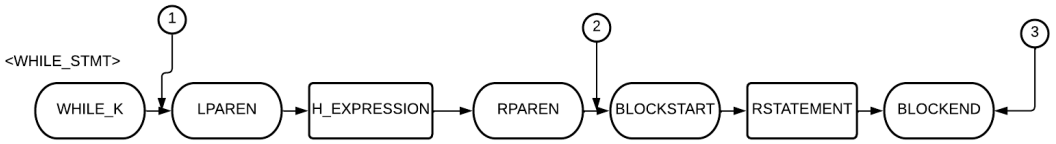
<NELSE>

ELSE

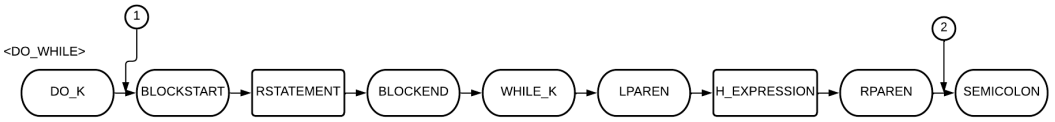
EMPTY



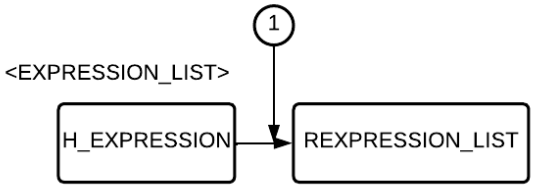
1. Genera un goto y mete el quadcounter al ultimo brinco



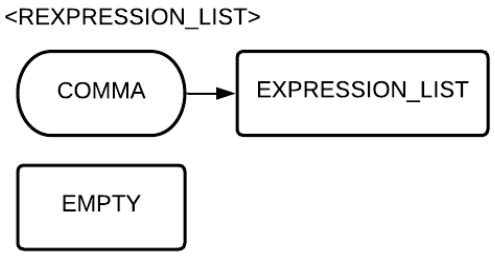
1. Push el counter a la blockstart
2. Revisa el tipo de la condicion, genera el quad y mete el quadcounter a la jumpstack
3. Genera el goto quad y mete el counter al ultimo brinco

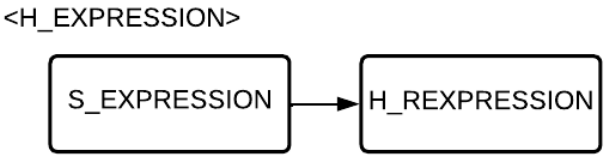


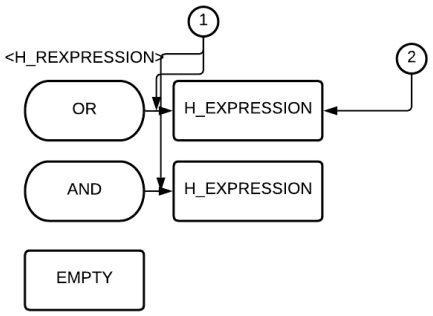
1. Mete el contador al jumpstack
2. Revisa el tipo de la condicion, genera el quad gotot



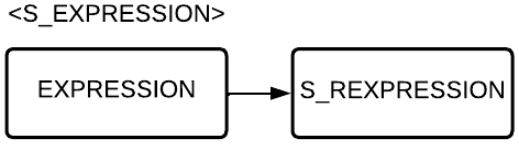
1. Creacion del cuadruplo parameter, para poder verificar los parametros en la maquina virtual, despues de validar su tipo y su orden

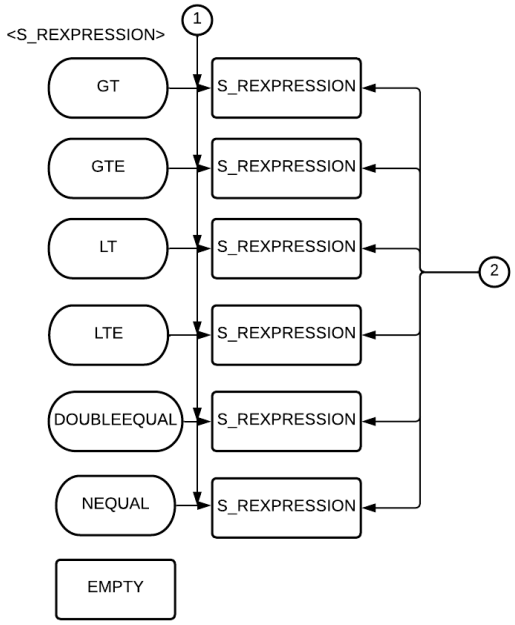




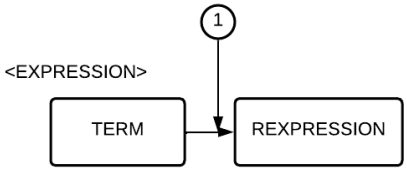


1. Agregar operadores al poper
2. Checa si el ultimo en el poper es || o && y en caso de que si, revisa el cubo semantico y genera el cuadruplo

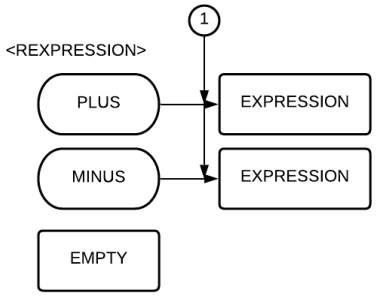




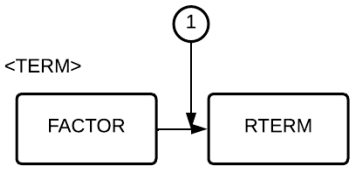
1. Agregar operadores al poper
2. Checa si el ultimo en el poper es > >= < <= == != y en caso de que si, revisa el cubo semantico y genera el cuadruplo



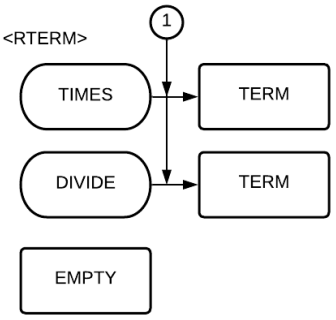
1. Checa si el ultimo en el poper es + - y en caso de que si, revisa el cubo semantico y genera el cuadruplo



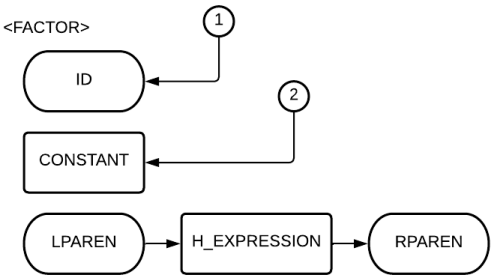
1. Agregar operadores al poper



1. Checa si el ultimo en el poper es * / y en caso de que si, revisa el cubo semantico y genera el cuadruplo



1. Mete el operador al stack poper



1. Revisa si la variable existe y si si agrega a la pila su tipo y operando
2. Revisa que tipo es la constante, lo mete al stack de tipos y mete el operando al stack de operandos y a si no esta, en la tabla de constantes

3.3- Administración de Memoria y Estructuras de Datos

Memoria virtual:

La administración de memoria está realizada principalmente por un objeto creado llamado Virtual_Memory el cual se encarga de la repartición de direcciones virtuales para la simulación de una memoria virtual real. El objeto cuenta con su constructor y dos funciones para realizar sus asignaciones.

```
class Virtual_Memory(object):
    def __init__(self):
        self.glob = [1000, 4000, 7000]
        self.local = [10000, 14000, 17000]
        self.temporal = [20000, 24000, 27000]
        self.constant = [30000, 34000, 37000]
```

Inicia con las direcciones virtuales base de cada scope, dependiendo del tipo de variable comienza en otro rango de dirección (int, float, string).

```
    #global 1000 - 9999
        #int    1000 - 3999
        #float  4000 - 6999
        #string 7000 - 9999
    #local  10000 - 19999
        #int    10000 - 13999
        #float  14000 - 16999
        #string 17000 - 19999
    #temporal 20000 - 29999
        #int    20000 - 23999
        #float  24000 - 26999
        #string 27000 - 29999
    #constants 30000 - 39999
        #int    30000 - 33999
        #float  34000 - 36999
        #string 37000 - 39999
```

Para comprender con más facilidad, este es el mapeo de las direcciones.

```
    def clear(self):
        self.local = [10000, 14000, 17000]
        self.temporal = [20000, 24000, 27000]
```

Esta función se manda a llamar cada vez que se inicia un nuevo scope, reinicia las direcciones virtuales locales y temporales para asignar correctamente.

```
    def assign(self, scope, type):
        if(scope == 'global'):
            if(type == 'int'):
                virtual_add = self.glob[0]
                self.glob[0] +=1
                return virtual_add
            elif(type == 'float'):
                virtual_add = self.glob[1]
                self.glob[1] +=1
                return virtual_add
            else:
                virtual_add = self.glob[2]
                self.glob[2] +=1
                return virtual_add
        if(scope == 'local'):
```

Este es un fragmento de la función que asigna, su propósito es asignar a cada variable su dirección virtual adecuada dependiendo de su scope y tipo y posteriormente le suma al contador para asignar la siguiente variable.

Directorio de funciones:

El directorio de funciones contiene todos los datos necesarios para almacenar valores, variables y atributos de cada función.

```
class FuncDirectory(object):
    def __init__(self):
        self.functions = {}
        self.program_name = "p"

    def decFunction(self, name, type, scope):
        self.functions[name] = {"type" : type, "scope" : scope, "name"
: name}
        self.functions[name]['var_table'] = [[], [], [], [[], [], []]]
        self.functions[name]['param_order'] = []
```

Esta es la estructura.
Primeramente cuando se declara una función se agrega al directorio su nombre, tipo y scope, después se le añade su tabla de variables que contiene múltiples listas en las que cada índice de lista pertenece a la misma variable para poder manejar y acceder fácilmente a sus elementos.
El primer índice de la tabla de variables es su identificador, el segundo el tipo de variable que es y el tercero es su dirección virtual.
El cuarto índice es una lista que contiene tres listas, su propósito es principalmente para las variables que son listas. El primer índice es para validar si es una lista, contiene la dirección de memoria base de una lista. El segundo índice es el tamaño de la lista y el tercer índice son las direcciones de memoria de todos los índices de la lista en orden.

Para comprender más fácilmente la estructura, ver la siguiente imagen.

```
#{
 # "name": {
 #   "type": "data"
 #   "scope": "data"
 #   "name": "data"
 #   "var_table": {
 #      "id": {
 #      }
#       "type":{
#       }
#       "virtualAddress":{
#       }
#       "arrays":{ dir0 = isArray
#                dir1 = size
#                dir2 = virtualAddIndexes
#       }
 #  }
 # }
#}
```

Cuádruplos:

El objeto de cuadruplos tiene como propósito crear los cuádruplos del programa escrito para así poder manejar su ejecución.

```
class quad(object):
    def __init__(self):
        self.quads = []
        self.poper = []
        self.op_stack = []
        self.type_stack = []
        self.jump_stack = []
        self.counter = 0
```

Este es su constructor, contiene pilas para la generación de sus cuadruplos correspondientes con su asociatividad adecuada.

El ‘quads’ es la lista en la que se le agrega cada cuádruplo generado en orden.
‘poper’ es la pila de las operaciones del lenguaje.
‘op_stack’ es la pila de los operandos o variables.

‘type_stack’ es la pila de tipos de los operandos, para validar con el cubo semántico las operaciones.
‘jump_stack’ es la pila de brincos para que la máquina virtual sepa a que cuádruplo se debe mover en el momento de hacer brincos.

```
    def generate_quad(self, op, left, right, t):
        q = [op, left, right, t]
        self.quads.append(q)
        self.counter += 1
```

Conforme va leyendo la gramática, utilizando puntos neurálgicos el compilador va metiendo y sacando en sus pilas sus operandos, operaciones, tipos, y brincos dependiendo de la asociatividad en la que lo va leyendo para crear cuadruplos y meterlos al stack de ‘quads’.

4.1- Descripción Máquina Virtual

La máquina virtual es un objeto que se encarga de la ejecución del programa, recibe los cuádruplos generados, el directorio de funciones y la tabla de constantes y en base a ellos realiza la ejecución.

```
class Virtual_Machine(object):
    def __init__(self):
        self.pointer = 0
        self.quads = []
        self.global_mem = Memory()
        self.local_mem = []
        self.temporal_mem = [Memory()]
        self.funcdir = None
        self.constant = []
        self.program_name = ''
        self.function_stack = []
        self.gosub_stack = []
        self.gosub_pointer_stack = []
        self.parameters_stack = []
        self.recursion_counter = 0
        self.pointer_array = [[], []]
```

Este es el constructor del objeto de la máquina virtual, antes de entrar a detalle a cada elemento se debe mencionar que utiliza el objeto memoria para el manejo de scopes:

```
class Memory(object):
    def __init__(self):
        self.int = [[], []]
        self.float = [[], []]
        self.string = [[], []]
```

El objeto de memoria simplemente contiene para cada tipo de variable una lista de dos listas en donde la primera lista es el valor de la variable y la segunda es su dirección virtual, comparten el mismo índice para poder acceder a ellos fácilmente.

El objeto de máquina virtual contiene funciones para su manejo:

```
def assign(self, virtual_address, value):
        #global int
        if(virtual_address >= 1000 and virtual_address <= 3999):
            index = self.global_mem.int[1].index(virtual_address)
            self.global_mem.int[0][index] = value
        #global float
        elif(virtual_address >= 4000 and virtual_address <= 6999):
            index = self.global_mem.float[1].index(virtual_address)
            self.global_mem.float[0][index] = value
```

La función ‘assign’ asigna el valor de una variable a su respectiva memoria en base a su scope y tipo.

```
   def register(self, virtual_address):

        #global int
        if(virtual_address >= 1000 and virtual_address <= 3999):
            #si no lo encuentra que lo registre
            if(virtual_address not in self.global_mem.int[1]):
                self.global_mem.int[1].append(virtual_address)
                self.global_mem.int[0].append(None)
        #global float
        elif(virtual_address >= 4000 and virtual_address <= 6999):
            #si no lo encuentra que lo registre
            if(virtual_address not in self.global_mem.float[1]):
                self.global_mem.float[1].append(virtual_address)
                self.global_mem.float[0].append(None)
```

La función ‘register’ abre espacio en la memoria de una dirección virtual en caso de que no se haya registrado anteriormente.

```
def find(self, virtual_address):

        #global int
        if(virtual_address >= 1000 and virtual_address <= 3999):
            index = self.global_mem.int[1].index(virtual_address)
            return self.global_mem.int[0][index]
        #global float
        elif(virtual_address >= 4000 and virtual_address <= 6999):
```

```
            index = self.global_mem.float[1].index(virtual_address)
            return self.global_mem.float[0][index]
```

La función ‘find’ busca en la memoria el valor real de una variable en base a su dirección virtual.

Regresando al objeto de la máquina virtual:

```
class Virtual_Machine(object):
    def __init__(self):
        self.pointer = 0
        self.quads = []
        self.global_mem = Memory()
        self.local_mem = []
        self.temporal_mem = [Memory()]
        self.funcdir = None
        self.constant = []
        self.program_name = ''
        self.function_stack = []
        self.gosub_stack = []
        self.gosub_pointer_stack = []
        self.parameters_stack = []
        self.recursion_counter = 0
        self.pointer_array = [[], []]
```

Los ‘quads’ son los cuádruplos importados que fueron hechos anteriormente.
El ‘pointer’ es simplemente el apuntador del índice de en que cuádruplo se está manejando.

```
    def start(self):
        while(self.pointer < len(self.quads)):
            self.do()
```

Utiliza esta función para navegar en cada cuádruplo y en base a qué operación tiene en el primer índice del cuádruple, genera su acción.

```
elif(operation == '+'):
            left_op = int(quad[1])
            right_op = int(quad[2])
            temporal = int(quad[3])

            if(left_op in self.pointer_array[0]):
                index = self.pointer_array[0].index(left_op)
                left_op = self.pointer_array[1][index]
            if(right_op in self.pointer_array[0]):
                index = self.pointer_array[0].index(right_op)
                right_op = self.pointer_array[1][index]

            self.register(left_op)
            self.register(right_op)
            self.register(temporal)

            result = self.find(left_op) + self.find(right_op)
            self.assign(temporal, result)
            self.pointer += 1
```

Por ejemplo digamos que tiene este cuadruplo : ["+", 1001, 30002, 20019]
Toma los elementos del cuádruplo, los registra, hace la operación y el resultado lo asigna a la variable indicada. (en un momento se explicara lo del arreglo de apuntadores)

```
self.global_mem = Memory()
```

La memoria global es un objeto de tipo Memory que contiene la información de la memoria global que es importada desde un inicio.

```
self.local_mem = []
```

La memoria local es una lista en la que cada vez que se entra a una función se le agrega un objeto de tipo Memory para poder manejar sus scopes

```
self.temporal_mem = [Memory()]
```

Lo mismo con la memoria temporal, solamente que ya comienza con una memoria añadida porque en el scope global también se utilizan variables temporales.

```
self.funcdir = None
```

Este es el directorio de funciones que fue importado.

```
self.constant = []
```
La tabla de constantes que fue importada.

```
self.program_name = ''
```
Se guarda el nombre del programa para poder hacer ciertas operaciones junto con el directorio de funciones.

```
self.function_stack = []
```
Esta es la pila de funciones, cada vez que se entra a una función nueva con la operación ‘ERA’ se añade el nombre para saber cuáles parámetros se van a importar.

```
self.gosub_stack = []
```
Es una pila para cada vez que se tiene la operación de ‘GOSUB’ con una función que tiene retorno se le agrega la dirección virtual de la función para que cuando llegue a la operación ‘RETURN’ haga un pop y le asigne el valor de retorno a su dirección.

```
self.gosub_stack = []
```
Es una pila para cuando cada vez que se tiene la operación de ‘GOSUB’ se le agrega a esta pila el cuádruplo siguiente (pointer + 1), para que cuando sea ‘RETURN’ o ‘ENDFUNC’ pueda hacer pop y regresar a donde se quedó anteriormente.

```
self.parameters_stack = []
```
Es una pila para cuando se manda a llamar una función y la operación es ‘PARAMETER’ le agrega a la pila sus parámetros y no perderlos porque en ‘ERA’ se abre otra instancia de memoria y no deja acceder a ellos. Se hace pop cuando sale de una función.

Cabe mencionar que para agregar a la pila los parámetros correctos utiliza esta función:

```
def find_parameters(self, virtual_address):
#local int
        elif(virtual_address >= 10000 and virtual_address <= 13999):
            index = self.local_mem[-2].int[1].index(virtual_address)
            return self.local_mem[-2].int[0][index]
        #local float
        elif(virtual_address >= 14000 and virtual_address <= 16999):
            index = self.local_mem[-2].float[1].index(virtual_address)
            return self.local_mem[-2].float[0][index]
```
(demuestra un fragmento de la función específicamente los locales para explicación)

Que es la misma funcionalidad que el método find() pero revisa en la penúltima memoria en vez de la última porque el ERA abre una memoria nueva vacía.

```
self.recursion_counter = 0
```

Simplemente un contador de en qué nivel de recursión se está metiendo y saber el índice de cuales pilas se deben acceder, por ejemplo con los parámetros. Que pueda importar los parámetros correctos de la función anterior en base al índice de este apuntador.

```
self.pointer_array = [[], []]
```

Esta lista es una simulación de un apuntador. Ayuda a acceder a los elementos de un arreglo, cuando la operación es un 'GET' el apuntador temporal es creado y se le apunta hacía el índice de memoria que tiene la casilla del arreglo especificada.

Regresando al ejemplo de la suma:

```
elif(operation == '+'):
            left_op = int(quad[1])
            right_op = int(quad[2])
            temporal = int(quad[3])

            if(left_op in self.pointer_array[0]):
                index = self.pointer_array[0].index(left_op)
                left_op = self.pointer_array[1][index]
            if(right_op in self.pointer_array[0]):
                index = self.pointer_array[0].index(right_op)
                right_op = self.pointer_array[1][index]

            self.register(left_op)
            self.register(right_op)
            self.register(temporal)

            result = self.find(left_op) + self.find(right_op)
            self.assign(temporal, result)
            self.pointer += 1
```

Se puede notar que en medio se tiene un if. Ese if revisa si el operando es un apuntador y si si lo es, le cambia su dirección por la dirección que está apuntando para poder hacer la operación correctamente.

5.1- Prueba Factorial - Iterativo y recursivo

Dato de entrada = 5

```
program factorial;

vars{
    int n;
    int result;
    }

function int recur(int i){
    vars{
    int x;
    int fact;
    }

    if(i == 0){
        return(1);
    }

    x = i - 1;
    fact = recur(x);
    fact = fact * i;
    return(fact);

}

function int iter(int i){
    vars{
    int x;
    int fact;
```

```
    }
    x = 1;
    fact = 1;

    while(x <= i){
        fact = fact * x;

        x = x + 1;
    }

    return(fact);

}

main(){
    n = 5;
    result = iter(n);
    write(result);
    result = recur(n);
    write(result);

}
```

Cuadruplos generados:

0 ['GOTO', '_', '_', 26]
1 ['==', 10000, 30000, 20000]
2 ['GOTOF', 20000, '_', 4]
3 ['RETURN', '_', '_', 30001]
4 ['-', 10000, 30001, 20001]
5 ['=', 20001, '_', 10001]
6 ['ERA', 3, '_', 'recur']
7 ['PARAMETER', 10001, '_', 0]
8 ['GOSUB', 'recur', '_', 1]
9 ['=', 1002, '_', 20002]
10 ['=', 20002, '_', 10002]
11 ['*', 10002, 10000, 20003]
12 ['=', 20003, '_', 10002]
13 ['RETURN', '_', '_', 10002]
14 ['ENDFunc', '_', '_', '_']
15 ['=', 30001, '_', 10001]
16 ['=', 30001, '_', 10002]
17 ['<=', 10001, 10000, 20000]
18 ['GOTOF', 20000, '_', 24]
19 ['*', 10002, 10001, 20001]
20 ['=', 20001, '_', 10002]
21 ['+', 10001, 30001, 20002]
22 ['=', 20002, '_', 10001]
23 ['GOTO', '_', '_', 17]
24 ['RETURN', '_', '_', 10002]
25 ['ENDFunc', '_', '_', '_']
26 ['=', 30002, '_', 1000]
27 ['ERA', 3, '_', 'iter']
28 ['PARAMETER', 1000, '_', 0]
29 ['GOSUB', 'iter', '_', 15]
30 ['=', 1003, '_', 20000]
31 ['=', 20000, '_', 1001]

32 ['WRITE', '_', '_', 1001]
33 ['ERA', 3, '_', 'recur']
34 ['PARAMETER', 1000, '_', 0]
35 ['GOSUB', 'recur', '_', 1]
36 ['=', 1002, '_', 20001]
37 ['=', 20001, '_', 1001]
38 ['WRITE', '_', '_', 1001]

Directorio de funciones:

{'factorial': {'type': 'program', 'scope': 'global', 'name': 'factorial', 'var_table': [['n', 'result', 'recur', 'iter'], ['int', 'int', 'int', 'int'], [1000, 1001, 1002, 1003], [[], [], []]], 'param_order': []}, 'recur': {'type': 'int', 'scope': 'local', 'name': 'recur', 'var_table': [['i', 'x', 'fact'], ['int', 'int', 'int'], [10000, 10001, 10002], [[], [], []]], 'param_order': ['int'], 'size': 3, 'start_add': 1}, 'iter': {'type': 'int', 'scope': 'local', 'name': 'iter', 'var_table': [['i', 'x', 'fact'], ['int', 'int', 'int'], [10000, 10001, 10002], [[], [], []]], 'param_order': ['int'], 'size': 3, 'start_add': 15}}

Resultado:

PS C:\Users\berny\OneDrive\Escritorio\COMP> & C:/Python310/python.exe c:/Users/berny/OneDrive/Escritorio/COMP/start.py
120
120

5.2- Prueba Fibonacci - Iterativo y recursivo

Dato de entrada: 12

```
program fibonacci;

vars{
    int uno;
    int dos;
    int tres;
    }

function int recur(int i){
    vars{
    int a;
    int b;
    int c;
    int d;
    int e;

}
    if(i <= 1){
        return(i);
    }

    d = i - 1;
    e = i - 2;
    a = recur(d);

    b = recur(e);

    c = a+b;
    return(c);
```

```
}

function int iter(int i){
    vars{
    int a;
    int b;
    int c;
    int d;
    int e;

}
    a = 0;
    b = 1;
    c = 0;
    d = 0;

    while(d < i){
        c = a + b;
        a = b;
        b = c;
        d = d + 1;
    }

    return(a);

}

main(){
    uno = 12;
    tres = iter(uno);
    write(tres);
    dos = recur(uno);
    write(dos);
```

}

Cuadruplos generados:

0 ['GOTO', '_', '_', 37]
1 ['<=', 10000, 30000, 20000]
2 ['GOTOF', 20000, '_', 4]
3 ['RETURN', '_', '_', 10000]
4 ['-', 10000, 30000, 20001]
5 ['=', 20001, '_', 10004]
6 ['-', 10000, 30001, 20002]
7 ['=', 20002, '_', 10005]
8 ['ERA', 6, '_', 'recur']
9 ['PARAMETER', 10004, '_', 0]
10 ['GOSUB', 'recur', '_', 1]
11 ['=', 1003, '_', 20003]
12 ['=', 20003, '_', 10001]
13 ['ERA', 6, '_', 'recur']
14 ['PARAMETER', 10005, '_', 0]
15 ['GOSUB', 'recur', '_', 1]
16 ['=', 1003, '_', 20004]
17 ['=', 20004, '_', 10002]
18 ['+', 10001, 10002, 20005]
19 ['=', 20005, '_', 10003]
20 ['RETURN', '_', '_', 10003]
21 ['ENDFunc', '_', '_', '_']
22 ['=', 30002, '_', 10001]
23 ['=', 30000, '_', 10002]
24 ['=', 30002, '_', 10003]
25 ['=', 30002, '_', 10004]
26 ['<', 10004, 10000, 20000]
27 ['GOTOF', 20000, '_', 35]
28 ['+', 10001, 10002, 20001]
29 ['=', 20001, '_', 10003]

30 ['=', 10002, '_', 10001]
31 ['=', 10003, '_', 10002]
32 ['+', 10004, 30000, 20002]
33 ['=', 20002, '_', 10004]
34 ['GOTO', '_', '_', 26]
35 ['RETURN', '_', '_', 10001]
36 ['ENDFunc', '_', '_', '_']
37 ['=', 30003, '_', 1000]
38 ['ERA', 6, '_', 'iter']
39 ['PARAMETER', 1000, '_', 0]
40 ['GOSUB', 'iter', '_', 22]
41 ['=', 1004, '_', 20000]
42 ['=', 20000, '_', 1002]
43 ['WRITE', '_', '_', 1002]
44 ['ERA', 6, '_', 'recur']
45 ['PARAMETER', 1000, '_', 0]
46 ['GOSUB', 'recur', '_', 1]
47 ['=', 1003, '_', 20001]
48 ['=', 20001, '_', 1001]
49 ['WRITE', '_', '_', 1001]

Directorio de funciones:

{'fibonacci': {'type': 'program', 'scope': 'global', 'name': 'fibonacci', 'var_table': [['uno', 'dos', 'tres', 'recur', 'iter'], ['int', 'int', 'int', 'int', 'i', 'c', 'd', 'e'], ['int', 'int', 'int', 'int', 'int', 'int'], [10000, 10001, 10002, 10003, 10004, 10005], [[], [], []]], 'param_order': ['int'], 'size': 6, 'start_add': 1}, 'iter': {'type': 'int', 'scope': 'local', 'name': 'iter', 'var_table': [['i', 'a', 'b', 'c', 'd', 'e'], ['int', 'int', 'int', 'int', 'int', 'int'], [10000, 10001, 10002, 10003, 10004, 10005], [[], [], []]], 'param_order': ['int'], 'size': 6, 'start_add': 22}}

Resultado:

PS C:\Users\berny\OneDrive\Escritorio\COMP> & C:/Python310/python.exe c:/Users/berny/OneDrive/Escritorio/COMP/start.py
144
144

5.3- Prueba Sort

Dato de entrada: [5, 1, 4, 2, 8]

```
program sort;

vars{
    array int list[5];
    int len;
    int x;
    int y;
    int z;
    int aux;
    int baux;
    int temp;
    }


main(){
    list[0] = 5;
    list[1] = 1;
    list[2] = 2;
    list[3] = 4;
    list[4] = 8;
    len = 5;
    x = 0;
    y = 0;

    while(x < len){
        z = len - x - 1;
        while(y < z){
            aux = list[y];
            baux = list[y+1];
```

```
            if(aux > baux){
                temp = list[y];
                list[y] = list[y+1];
                list[y+1] = temp;
            }
            y = y + 1;
        }
        x = x + 1;
    }
    write(list[0]);
    write(list[1]);
    write(list[2]);
    write(list[3]);
    write(list[4]);

}
```

Cuadruplos generados:

0 ['GOTO', '_', '_', 1]
1 ['GET', 30001, 1000, 20000]
2 ['=', 30000, '_', 20000]
3 ['GET', 30002, 1000, 20001]
4 ['=', 30002, '_', 20001]
5 ['GET', 30003, 1000, 20002]
6 ['=', 30003, '_', 20002]
7 ['GET', 30004, 1000, 20003]
8 ['=', 30005, '_', 20003]
9 ['GET', 30005, 1000, 20004]
10 ['=', 30006, '_', 20004]
11 ['=', 30000, '_', 1005]
12 ['=', 30001, '_', 1006]
13 ['=', 30001, '_', 1007]

14 ['<', 1006, 1005, 20005]
15 ['GOTOF', 20005, '_', 43]
16 ['-', 1005, 1006, 20006]
17 ['-', 20006, 30002, 20007]
18 ['=', 20007, '_', 1008]
19 ['<', 1007, 1008, 20008]
20 ['GOTOF', 20008, '_', 40]
21 ['GET', 1007, 1000, 20009]
22 ['=', 20009, '_', 1009]
23 ['+', 1007, 30002, 20010]
24 ['GET', 20010, 1000, 20011]
25 ['=', 20011, '_', 1010]
26 ['>', 1009, 1010, 20012]
27 ['GOTOF', 20012, '_', 37]
28 ['GET', 1007, 1000, 20013]
29 ['=', 20013, '_', 1011]
30 ['GET', 1007, 1000, 20014]
31 ['+', 1007, 30002, 20015]
32 ['GET', 20015, 1000, 20016]
33 ['=', 20016, '_', 20014]
34 ['+', 1007, 30002, 20017]
35 ['GET', 20017, 1000, 20018]
36 ['=', 1011, '_', 20018]
37 ['+', 1007, 30002, 20019]
38 ['=', 20019, '_', 1007]
39 ['GOTO', '_', '_', 19]
40 ['+', 1006, 30002, 20020]
41 ['=', 20020, '_', 1006]
42 ['GOTO', '_', '_', 14]
43 ['GET', 30001, 1000, 20021]
45 ['GET', 30002, 1000, 20022]
46 ['WRITE', '_', '_', 20022]
47 ['GET', 30003, 1000, 20023]
48 ['WRITE', '_', '_', 20023]

49 ['GET', 30004, 1000, 20024]

50 ['WRITE', '_', '_', 20024]

51 ['GET', 30005, 1000, 20025]

52 ['WRITE', '_', '_', 20025]

Directorio de funciones:

{'sort': {'type': 'program', 'scope': 'global', 'name': 'sort', 'var_table': [['list', 'len', 'x', 'y', 'z', 'aux', 'baux', 'temp'], ['int', 'int', 'int', 'int', 'int', 'int', 'int', 'int'], [1000, 1005, 1006, 1007, 1008, 1009, 1010, 1011], [[1000], [5], [[1000, 1001, 1002, 1003, 1004]]]], 'param_order': []}}

Resultado:

PS C:\Users\berny\OneDrive\Escritorio\COMP> & C:/Python310/python.exe c:/Users/berny/OneDrive/Escritorio/COMP/start.py

1

2

4

5

8

5.4- Prueba Find

Dato de entrada: 2

```
program find;

vars{
    int x;
    int mid;
    int aux;
    int auxx;
    array int list[10];
    int liminf;
    int limsup;
    }


main(){
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;
    list[3] = 4;
    list[4] = 5;
    list[5] = 6;
    list[6] = 7;
    list[7] = 8;
    list[8] = 9;
    list[9] = 10;
    liminf = 0;
    limsup = 9;
    mid = liminf + limsup;
    mid = mid / 2;
```

```
    x = 2;

    aux = list[mid];
    while(limsup - liminf > 1){

        mid = liminf + limsup;
        mid = mid / 2;

        aux = list[mid];
        if(aux < x){
            liminf = mid + 1;
        }
        else{
            limsup = mid;
        }
    }

    aux = list[liminf];

    if(aux == x){
        write(liminf);
    }

    aux = list[limsup];

    if(aux == x){
        write(limsup);
    }


}
```

Cuadruplos generados:

0 ['GOTO', '_', '_', 1]

1 ['GET', 30001, 1004, 20000]

2 ['=', 30002, '_', 20000]

3 ['GET', 30002, 1004, 20001]

4 ['=', 30003, '_', 20001]

5 ['GET', 30003, 1004, 20002]

6 ['=', 30004, '_', 20002]

7 ['GET', 30004, 1004, 20003]

8 ['=', 30005, '_', 20003]

9 ['GET', 30005, 1004, 20004]

10 ['=', 30006, '_', 20004]

11 ['GET', 30006, 1004, 20005]

12 ['=', 30007, '_', 20005]

13 ['GET', 30007, 1004, 20006]

14 ['=', 30008, '_', 20006]

15 ['GET', 30008, 1004, 20007]

16 ['=', 30009, '_', 20007]

17 ['GET', 30009, 1004, 20008]

18 ['=', 30010, '_', 20008]

19 ['GET', 30010, 1004, 20009]

20 ['=', 30000, '_', 20009]

21 ['=', 30001, '_', 1014]

22 ['=', 30010, '_', 1015]

23 ['+', 1014, 1015, 20010]

24 ['=', 20010, '_', 1001]

25 ['/', 1001, 30003, 20011]

26 ['=', 20011, '_', 1001]

27 ['=', 30003, '_', 1000]

28 ['GET', 1001, 1004, 20012]

29 ['=', 20012, '_', 1002]

30 ['-', 1015, 1014, 20013]

31 ['>', 20013, 30002, 20014]

32 ['GOTOF', 20014, '_', 46]

33 ['+', 1014, 1015, 20015]
34 ['=', 20015, '_', 1001]
35 ['/', 1001, 30003, 20016]
36 ['=', 20016, '_', 1001]
37 ['GET', 1001, 1004, 20017]
38 ['=', 20017, '_', 1002]
39 ['<', 1002, 1000, 20018]
40 ['GOTOF', 20018, '_', 44]
41 ['+', 1001, 30002, 20019]
42 ['=', 20019, '_', 1014]
43 ['GOTO', '_', '_', 45]
44 ['=', 1001, '_', 1015]
45 ['GOTO', '_', '_', 30]
46 ['GET', 1014, 1004, 20020]
47 ['=', 20020, '_', 1002]
49 ['GOTOF', 20021, '_', 51]
50 ['WRITE', '_', '_', 1014]
51 ['GET', 1015, 1004, 20022]
52 ['=', 20022, '_', 1002]
53 ['==', 1002, 1000, 20023]
54 ['GOTOF', 20023, '_', 56]
55 ['WRITE', '_', '_', 1015]

Directorio de funciones:

{'find': {'type': 'program', 'scope': 'global', 'name': 'find', 'var_table': [['x', 'mid', 'aux', 'auxx', 'list', 'liminf', 'limsup'], ['int', 'int', 'int', 'int', 'int', 'int', 'int'], [1000, 1001, 1002, 1003, 1004, 1014, 1015], [[1004], [10], [[1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013]]]], 'param_order': []}}

Resultado(índice del arreglo):

PS C:\Users\berny\OneDrive\Escritorio\COMP> & C:/Python310/python.exe c:/Users/berny/OneDrive/Escritorio/COMP/start.py
1

5.4- Prueba Estadística y Gráfica

Dato de entrada: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10] [2, 4, 4, 6, 8, 10, 12, 14, 16, 18, 20]

El programa saca el promedio, el mediano, el modo de las dos listas y realiza una gráfica con los dos datos de entrada.

```
program stat;

vars{
    array int one[11];
    array int two[11];
    int average;
    int med;
    int repeat;
    }


main(){
    one[0] = 1;
    one[1] = 2;
    one[2] = 3;
    one[3] = 4;
    one[4] = 5;
    one[5] = 6;
    one[6] = 7;
    one[7] = 8;
    one[8] = 9;
    one[9] = 10;
    one[10] = 10;

    two[0] = 2;
    two[1] = 4;
```

```
    two[2] = 4;
    two[3] = 6;
    two[4] = 8;
    two[5] = 10;
    two[6] = 12;
    two[7] = 14;
    two[8] = 16;
    two[9] = 18;
    two[10] = 20;

    average = mean(one);
    write(average);
    average = mean(two);
    write(average);

    med = median(one);
    write(med);
    med = median(two);
    write(med);

    repeat = mode(one);
    write(repeat);
    repeat = mode(two);
    write(repeat);

    graph(one, two);


}
```

Cuadruplos generados:

```
0 ['GOTO', '_', '_', 1]
1 ['GET', 30001, 1004, 20000]
2 ['=', 30002, '_', 20000]
```

3 ['GET', 30002, 1004, 20001]
4 ['=', 30003, '_', 20001]
5 ['GET', 30003, 1004, 20002]
6 ['=', 30004, '_', 20002]
7 ['GET', 30004, 1004, 20003]
8 ['=', 30005, '_', 20003]
9 ['GET', 30005, 1004, 20004]
10 ['=', 30006, '_', 20004]
11 ['GET', 30006, 1004, 20005]
12 ['=', 30007, '_', 20005]
13 ['GET', 30007, 1004, 20006]
14 ['=', 30008, '_', 20006]
15 ['GET', 30008, 1004, 20007]
16 ['=', 30009, '_', 20007]
17 ['GET', 30009, 1004, 20008]
18 ['=', 30010, '_', 20008]
19 ['GET', 30010, 1004, 20009]
20 ['=', 30000, '_', 20009]
21 ['=', 30001, '_', 1014]
22 ['=', 30010, '_', 1015]
23 ['+', 1014, 1015, 20010]
24 ['=', 20010, '_', 1001]
25 ['/', 1001, 30003, 20011]
26 ['=', 20011, '_', 1001]
27 ['=', 30003, '_', 1000]
28 ['GET', 1001, 1004, 20012]
29 ['=', 20012, '_', 1002]
30 ['-', 1015, 1014, 20013]
31 ['>', 20013, 30002, 20014]
32 ['GOTOF', 20014, '_', 46]
33 ['+', 1014, 1015, 20015]
34 ['=', 20015, '_', 1001]
35 ['/', 1001, 30003, 20016]
36 ['=', 20016, '_', 1001]

37 ['GET', 1001, 1004, 20017]
38 ['=', 20017, '_', 1002]
39 ['<', 1002, 1000, 20018]
40 ['GOTOF', 20018, '_', 44]
41 ['+', 1001, 30002, 20019]
42 ['=', 20019, '_', 1014]
43 ['GOTO', '_', '_', 45]
44 ['=', 1001, '_', 1015]
45 ['GOTO', '_', '_', 30]
46 ['GET', 1014, 1004, 20020]
47 ['=', 20020, '_', 1002]
49 ['GOTOF', 20021, '_', 51]
50 ['WRITE', '_', '_', 1014]
51 ['GET', 1015, 1004, 20022]
52 ['=', 20022, '_', 1002]
53 ['==', 1002, 1000, 20023]
54 ['GOTOF', 20023, '_', 56]
55 ['WRITE', '_', '_', 1015]

Directorio de funciones:

{'find': {'type': 'program', 'scope': 'global', 'name': 'find', 'var_table': [['x', 'mid', 'aux', 'auxx', 'list', 'liminf', 'limsup'], ['int', 'int', 'int', 'int', 'int', 'int', 'int'], [1000, 1001, 1002, 1003, 1004, 1014, 1015], [[1004], [10], [[1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013]]]], 'param_order': []}}

Resultado:

PS C:\Users\berny\OneDrive\Escritorio\COMP> & C:/Python310/python.exe c:/Users/berny/OneDrive/Escritorio/COMP/start.py
5.909090909090909
10.363636363636363
6
10
10
4

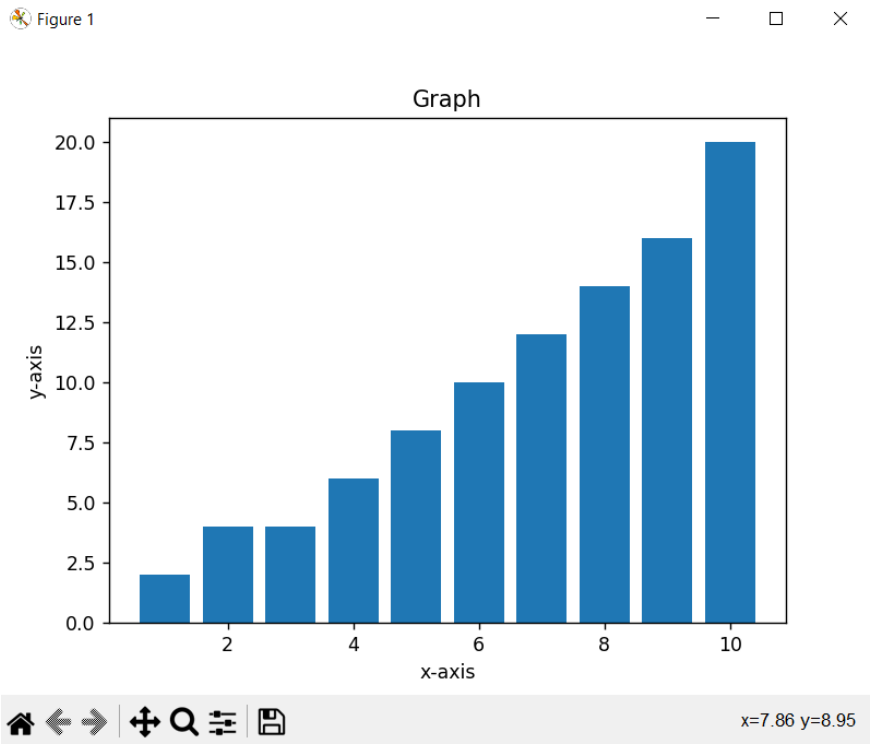


Figure 1
Graph
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
y-axis
2
4
6
8
10
x-axis
x=7.86 y=8.95