Tecnológico de Monterrey

Campus Monterrey

Compiladores

TC 3040

**MARFLE**

Autores

Rubén Alejandro Valdez Ruiz Ana Eline Guajardo Mozo  
A00803015 A01033641  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8 de mayo de 2013

**Índice**

Documentación

[1. Descripción del proyecto 3](#_Toc355688755)

[1.1 Visión, Objetivos y Alcance del Proyecto 3](#_Toc355688756)

[1.2 Análisis de Requerimientos y Casos de Uso Generales 3](#_Toc355688757)

[1.3 Descripción de los principales *Test Cases* 11](#_Toc355688758)

[1.4 Descripción del proceso 11](#_Toc355688759)

[2. Descripción del Lenguaje 11](#_Toc355688760)

[2.1 Nombre del lenguaje 11](#_Toc355688761)

[2.2 Descripción genérica 11](#_Toc355688762)

[2.3 Descripción de errores 11](#_Toc355688763)

[3. Descripción del Compilador 11](#_Toc355688764)

[4. Descripción de la Máquina Virtual 11](#_Toc355688765)

[4.1 General 11](#_Toc355688766)

[4.2 Arquitectura 11](#_Toc355688767)

[5. Pruebas del Funcionamiento del Lenguaje 11](#_Toc355688768)

[6. Listados del Proyecto 11](#_Toc355688769)

Documentación

1. Descripción del proyecto
   1. Visión, Objetivos y Alcance del Proyecto
      1. **Visión**

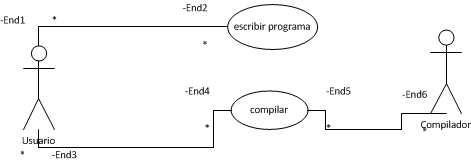
Para el año 2020, se busca que el lenguaje esté al nivel de Visual Basic .NET™ y Python™ como el lenguaje ideal para un principiante, es decir, que sea la primera opción para aprender a programar.

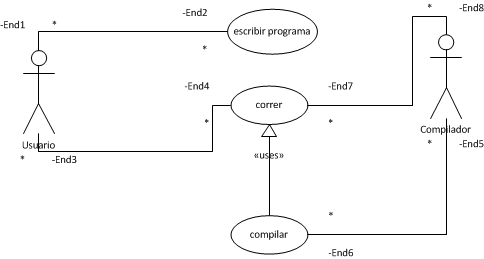
* + 1. **Objetivo del lenguaje**

Se busca que el lenguaje Marfle sea un buen comienzo para aquellos que están empezando a programar en lenguajes orientado a objetos, como es el caso con Python, para que cuando el usuario se decida por aprender lenguajes más avanzados como C++ ya tenga un conocimiento básico de cómo es programar, y que no lo sienta tan abrumador.

* + 1. **Alcance**Como ya se mencionó, la intención es que Marfle sea una lenguaje orientado a objetos básico, mediante el cual se pueden realizar problemas sencillos con la intención de ayudar en el aprendizaje de un programador primerizo, como lo son conceptos básicos como operaciones aritméticas, utilizar arreglos, crear objetos, etc.
  1. Análisis de Requerimientos y Casos de Uso Generales

Dado a que es un LOO, nuestro lenguaje debe permitir la definición de clases con atributos y métodos, que maneje herencia, recursión, estructuras de datos como arreglos y matrices, ciclos como el while, do while y el for, y un estatuto de condición como el if.

* + 1. **Casos de Uso  
       **



* 1. Descripción de los principales *Test Cases*
* Fibonacci iterativo: resolver de manera iterativa el famoso algoritmo de Fibonacci.
* Factorial iterativo: obtener el factorial de cualquier número entero utilizando ciclos.
  1. Descripción del proceso

Se utilizó la herramienta GitHub para el manejo de versiones del compilador. Asimismo, se mantuvo una bitácora que se actualizaba al menos una vez por semana, en la que mencionaba en cada nueva entrada una breve descripción de cambios o avances significativos que se hayan hecho en el código.

* + 1. **Bitácoras**

Jueves 7 Marzo 2013

* Se agregó la palabra reservada 'obj' al lenguaje para la declaración de variables de tipo objeto.

Jueves 14 de Marzo 2013

* Se tienen los cuádruplos de las operaciones básicas, el estatuto "return" y el estatuto "if" sin la parte del ELSE

Domingo 7 Abril de 2013

* Se codificó el cubo semántico

Domingo 14 Abril de 2013

* Se cambió la estructura a usar para los módulos y objetos: Se reemplazaron las tuplas por objetos de tipo diccionario, para tener una mucho mejor indexación de elementos
* Se implementó completamente el cubo semántico
* Inicio de planeación de memorias

Lunes 15 Abril de 2013

* Se implementaron los módulos y variables de objeto. Ya se generan cuádruplos de métodos de objeto.
* Se necesita ver la implementación de asignación de valores a variables de objeto antes de la declaración de módulos de objeto y hacer que se pueda hacer lo mismo desde el main del programa.

Martes 16 Abril de 2013

* Se implementó el ciclo FOR en el lenguaje.
* Se valida que la declaración de una variable de tipo objeto tenga en existencia el objeto.
* Se introdujo la parametrización de métodos.

Miércoles 17 Abril de 2013

* Se implementó el ERA en los módulos y se crea el cuádruplo correspondiente, así como los cuádruplos de parámetros donde son necesarios.

Viernes 19 Abril de 2013

* Se implementó la declaración de arreglos.
* Se implementó el retorno de módulo, así como el cuádruplo de GOSUB.

Domingo 21 de Abril de 2013

* Se inició la máquina virtual. Hace sumas de constantes enteras y temporales enteras, pero tiene problemas al asignarlas a variables globales.
  + 1. **Reflexiones**

*Ana*: Creo que éste ha sido fácilmente el proyecto más grande y pesado que he tenido durante toda mi carrera. Exigió tiempo y recursos, además de aprender un nuevo lenguaje; y a pesar de tratar de ir al corriente lo mejor posible, me quitó también varias horas de sueño. De ahora en adelante lo pensaré dos veces antes de usar un arreglo al programar.

*Rubén*: Para el proyecto de Compiladores de Ene-Mayo de 2013 aprendí a utilizar a Python, y la herramienta de gramáticas PLY. Ha sido una experiencia interesante en el sentido que ahora sé usar un lenguaje nuevo así como tengo nuevos conocimientos sobre qué hace una computadora internamente al escribir código para mis programas. El mayor reto de esta materia para mí ha sido el poder organizarme para completar todas las entregas en tiempo y forma.

1. Descripción del Lenguaje
   1. Nombre del lenguaje

Marfle

* 1. Descripción genérica

Marfle es un Lenguaje Orientado a Objetos que cuenta con los requerimientos básicos de cualquier lenguaje OO; se pueden hacer operaciones aritméticas, manejar objetos, crear métodos, etc.

* 1. Descripción de errores
* Errores de sintaxis (que no se siga el sintaxis establecido por el manual).
* Variables y objetos no declarados.
* Volver a declarar una variable ya inicializada.
* Tipos inválidos al momento de asociar valores a variables.
* Referirse a variables u objetos no declarados.
* Overflow de la memoria.
* Estructura de paréntesis.

1. Descripción del Compilador

El compilador fue escrito en Python, utilizando PLY, una implementación de lex y yacc para Python.

* 1. Análisis léxico

Identificadores: [a-z][A-Za-z0-9\_]\*

Constantes enteras: [0-9]+

Constantes flotantes: [0-9]+.[0-9]+

Constante string: \”(Identificadores | Const. Entera | Const. Flotante)\*\”

Constantes booleana: (true | false)

Tokens:

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

t\_LCORCH = r'\['

t\_RCORCH = r'\]'

t\_LBRACK = r'\{'

t\_RBRACK = r'\}'

t\_MAS = r'\+'

t\_MENOS = r'\-'

t\_POR = r'\\*'

t\_DIVISI = r'\/'

t\_IGUAL = r'\='

t\_MENQUE = r'\<'

t\_MAYQUE = r'\>'

t\_MENIGU = r'\<\='

t\_MAYIGU = r'\>\='

t\_DIFERE = r'\!\='

t\_IGUIGU = r'\=\='

t\_AND = r'\&\&'

t\_OR = r'\|\|'

t\_NOT = r'\!'

t\_PUNTO = r'\.'

t\_COMMA = r'\,'

t\_SEMCOL = r'\;'

t\_CONSTR = r'\"[A-Za-z0-9 ]+\"'

* 1. Análisis de Sintaxis

PROGRAMA

MAIN

OBJETO

BLOQUE

ESTATUTO

VARS

TAMAÑO

RETURN

ASIGNACION

ESCRITURA

LECTURA

CONDICION – IF

CICLO

DO WHILE

WHILE

FOR

DIMENSION

TÉRMINO

FACTOR

MÓDULO

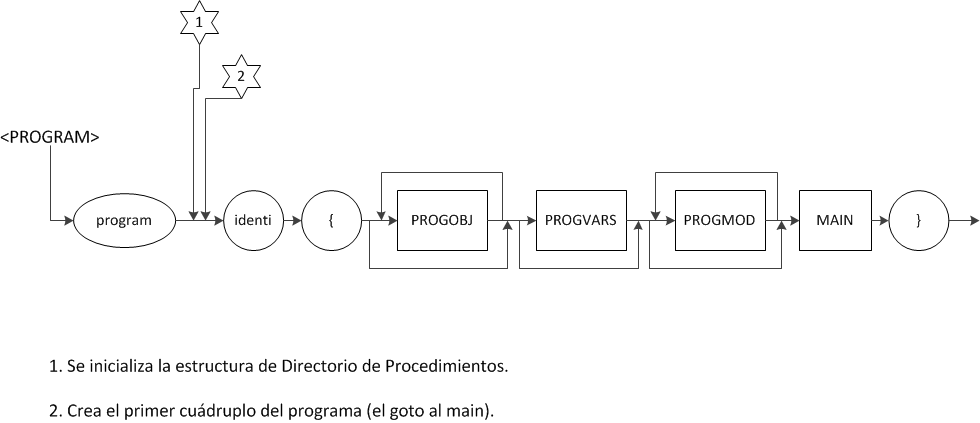
VARCTE

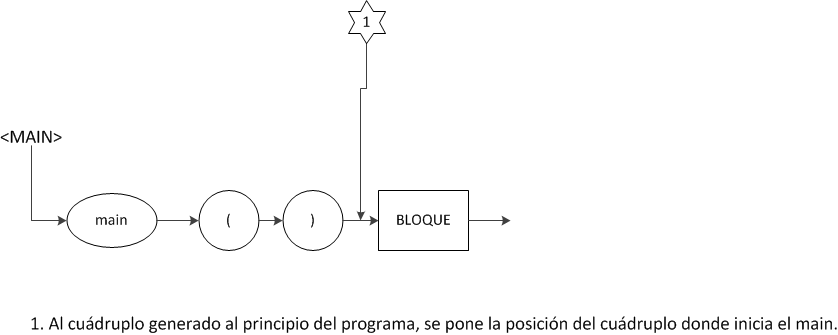
TIPO

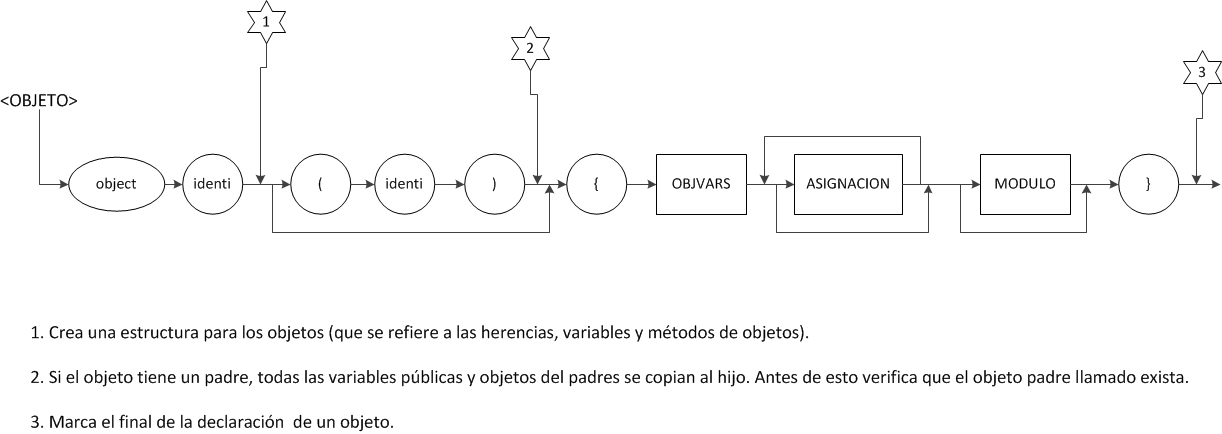
EXPRESIÓN

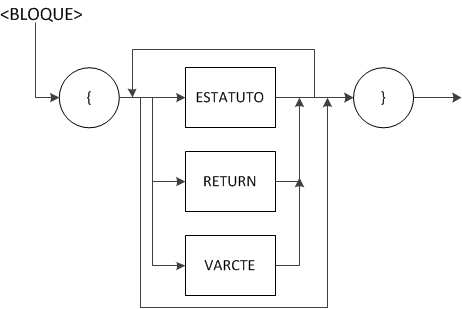
SUPEREXPRESIÓN

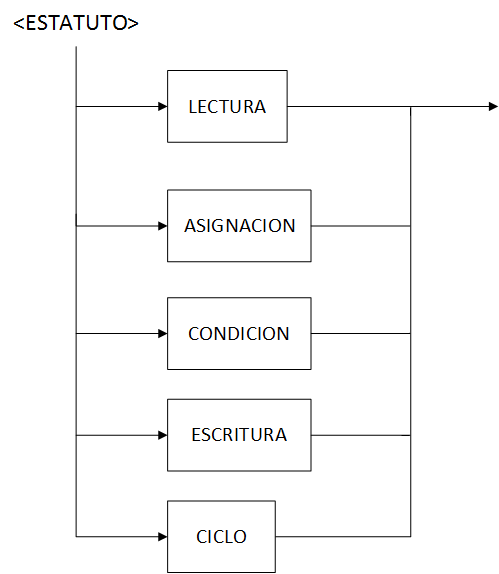
* 1. Análisis semántico

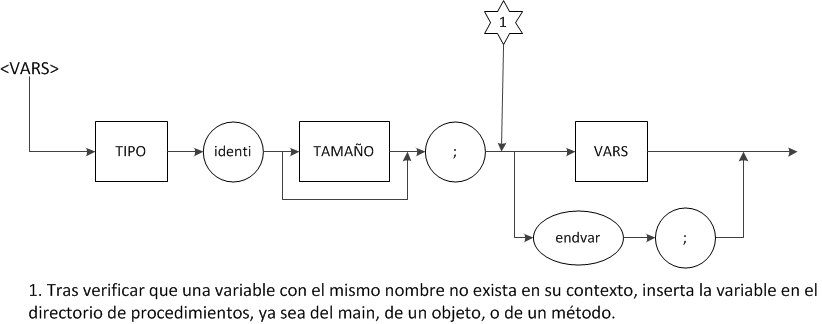


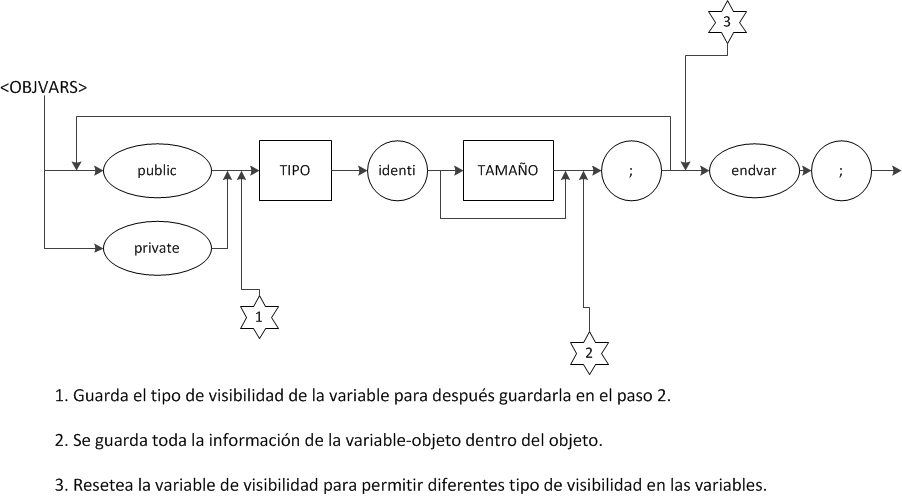


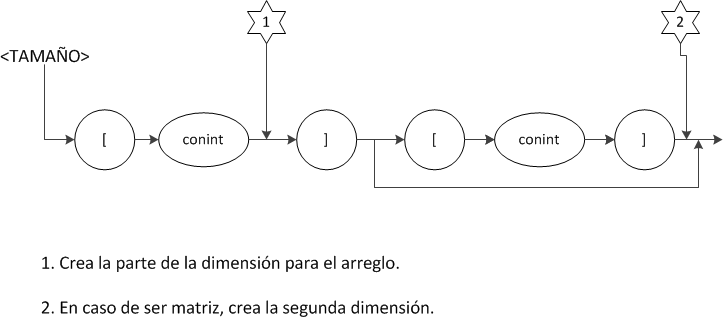


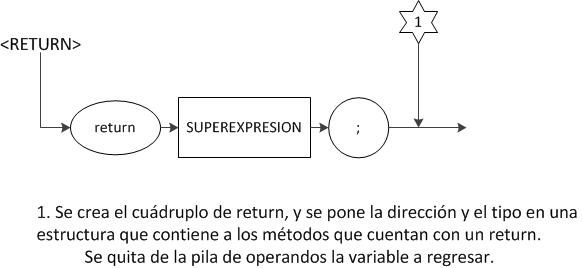


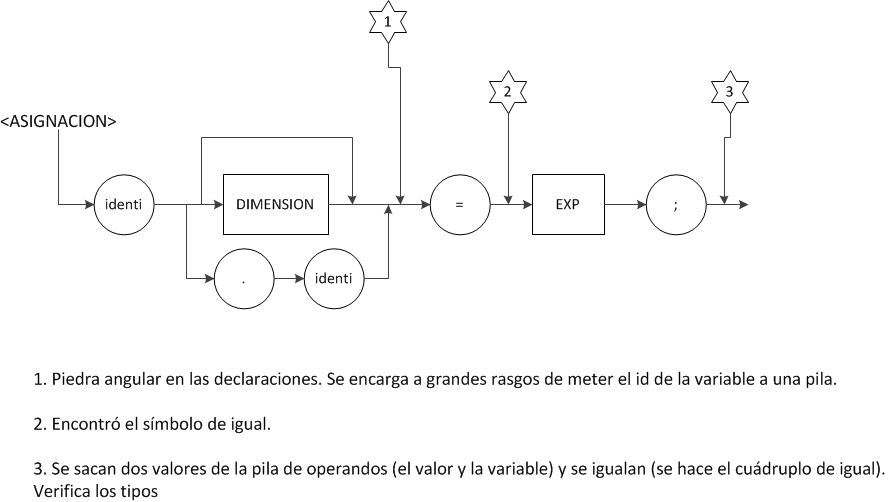


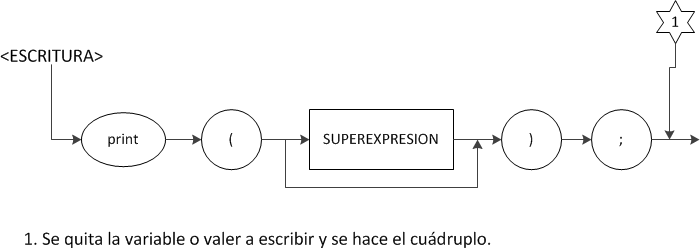


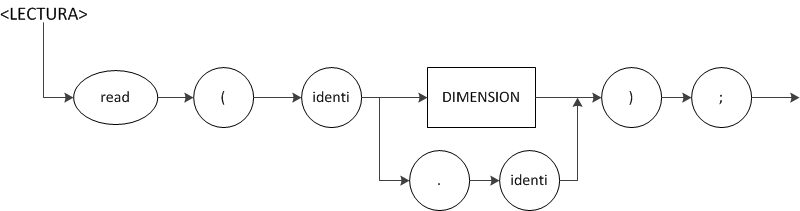


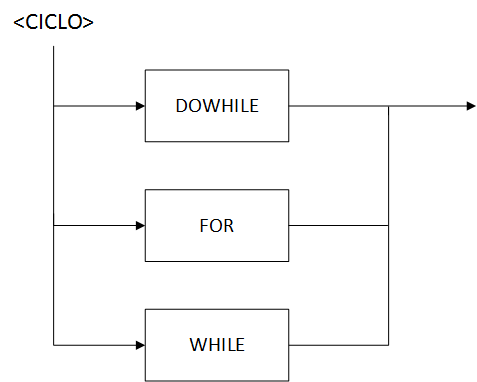


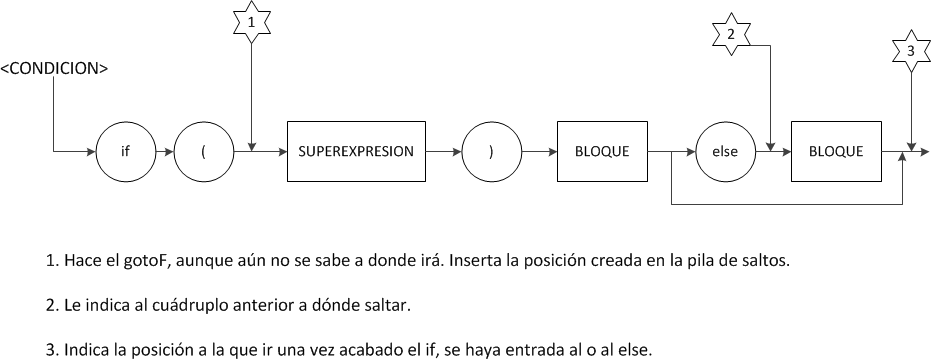


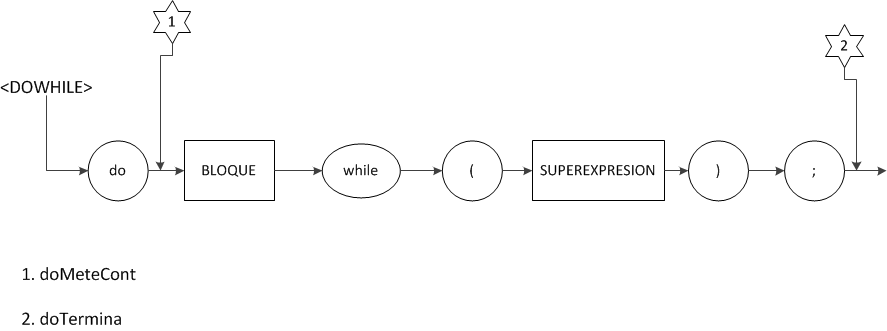


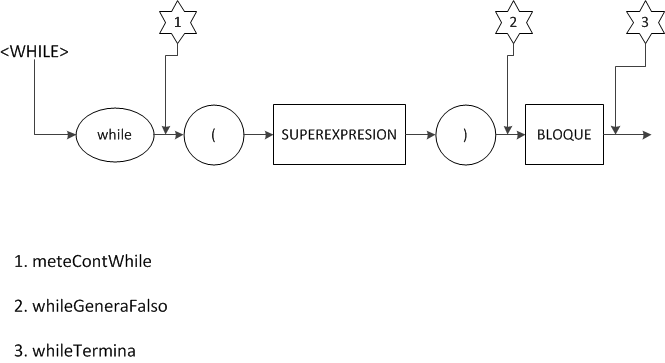


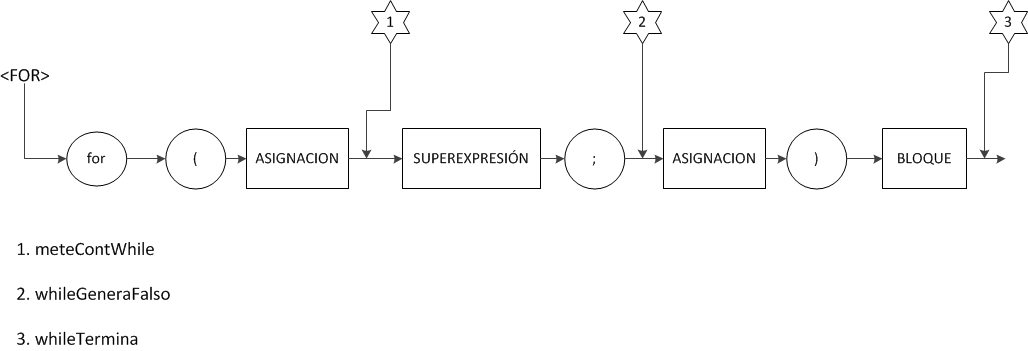


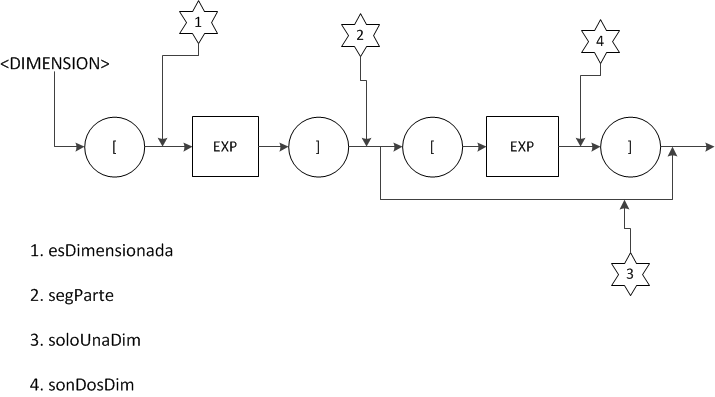


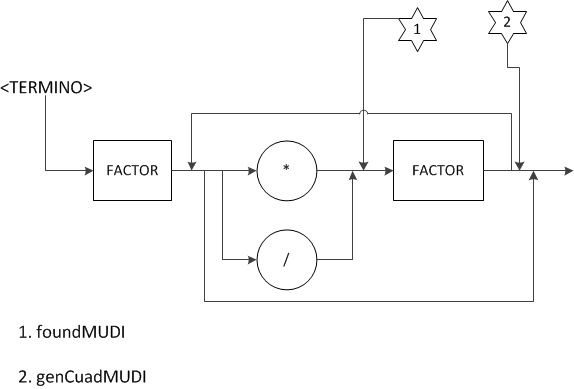
****

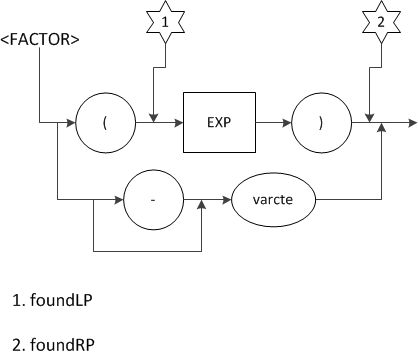
****

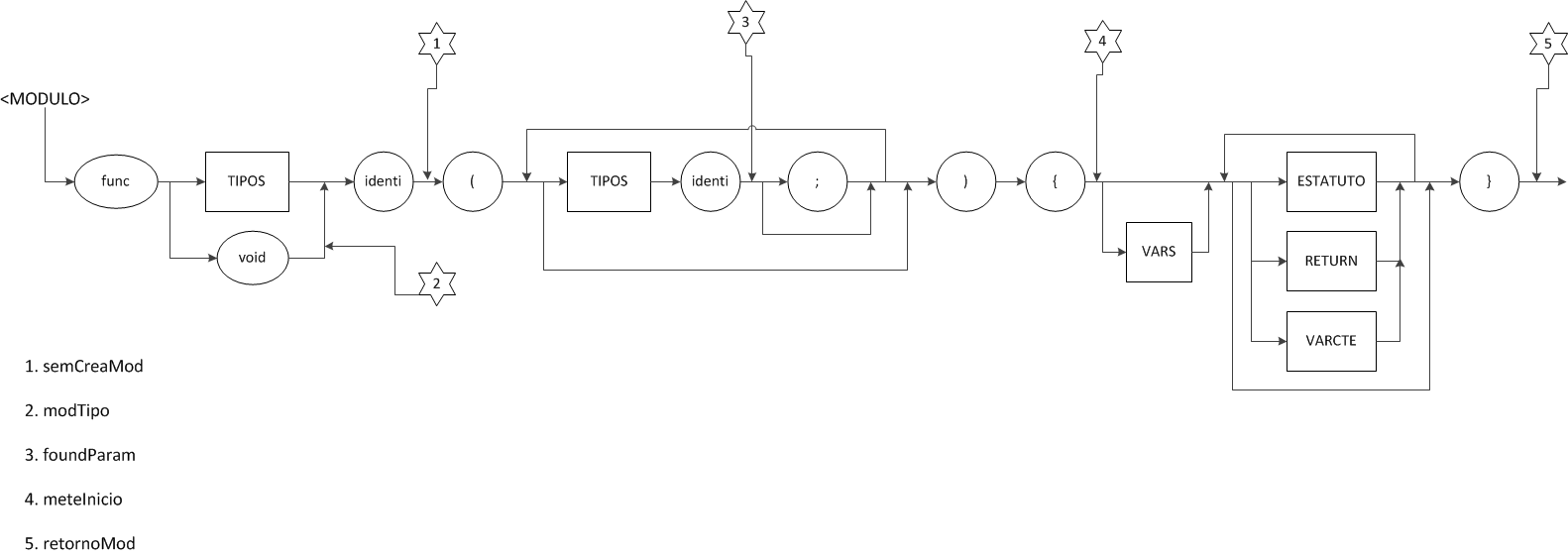
****

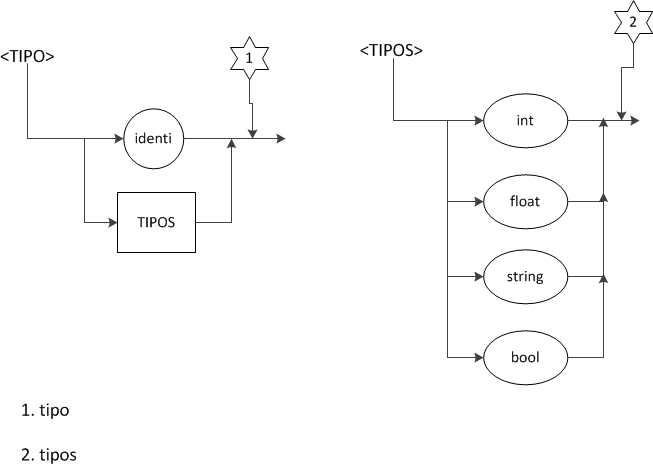
****

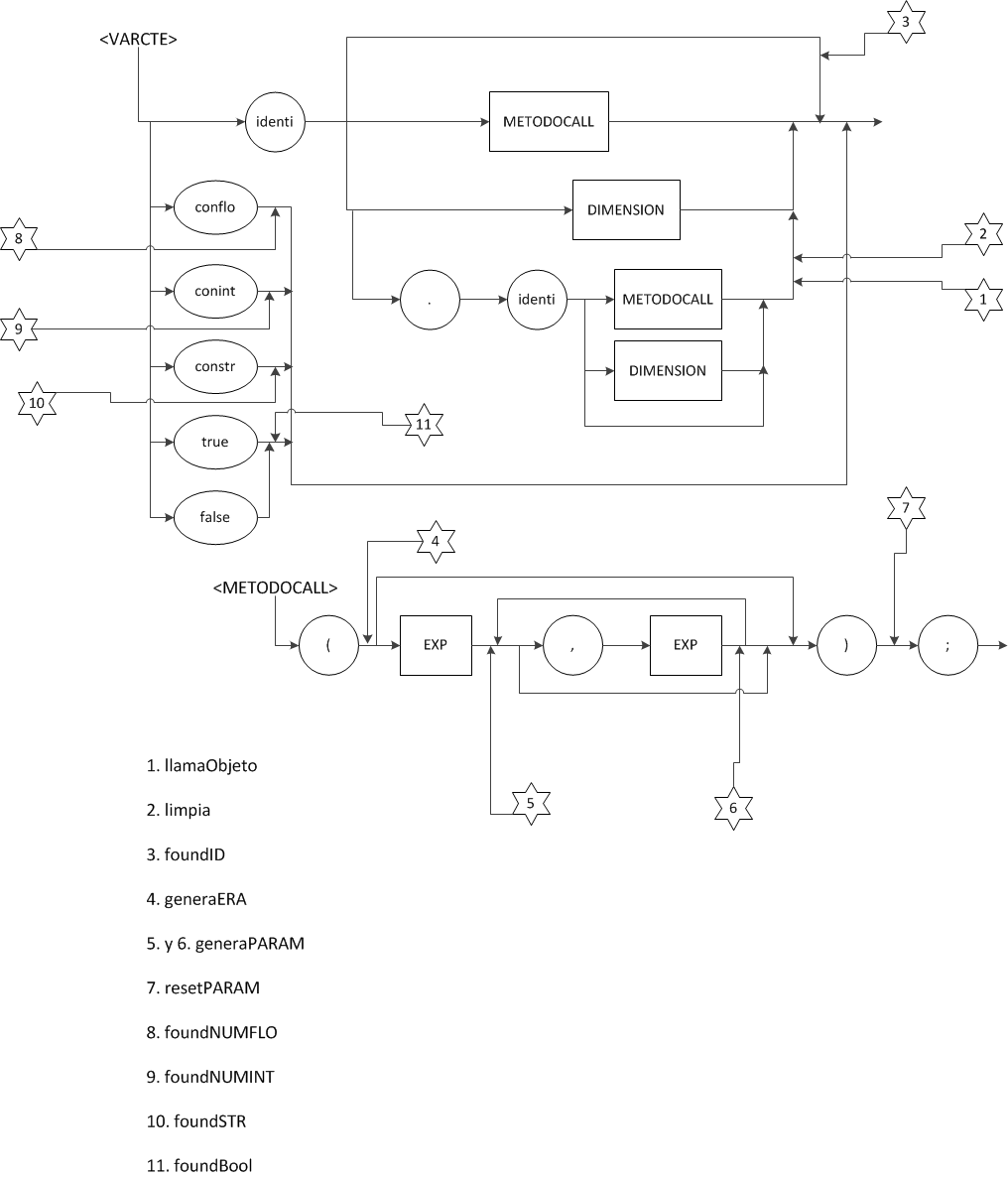
****

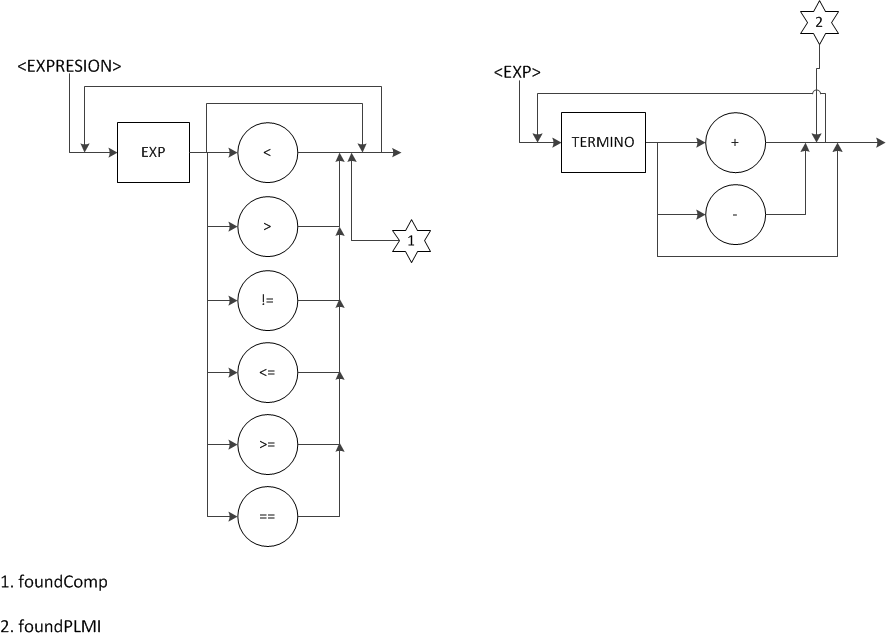
****

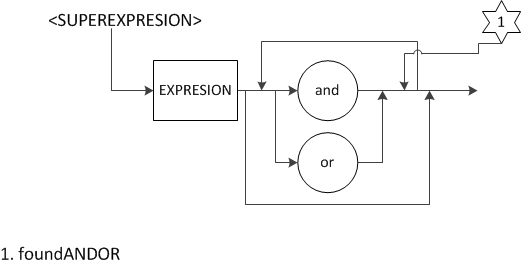
****

****

****

****

****

****

1. Descripción de la Máquina Virtual
   1. General

En su mayoría, la máquina virtual fue escrita en Linux, utilizando nuevamente Python.

* 1. Arquitectura

1. Pruebas del Funcionamiento del Lenguaje
   1. Testcase 1

program uno {

object derp

{

public int x;

public float barf;

public int obArr[3];

public bool bolly;

public string stringy;

private int ark;

endvar;

func void holis()

{

int a;

endvar;

a = 2;

print(a);

}

}

int a;

int arr[5];

int dosie[4][5];

float arr2[4];

int b;

float d;

derp wow;

derp wii;

string str;

bool boolean;

endvar;

func int fibonacci (int n)

{

int prevPrev;

int prev;

int result;

int i;

endvar;

prevPrev = 0;

prev = 1;

result = 0;

i = 0;

if (n == 0 || n == 1)

{

result = n;

}

else

{

for (i = 0; i < n; i = i + 1;)

{

result = prev + prevPrev;

prevPrev = prev;

prev = result;

}

}

return(result);

}

func void busqueda(int n)

{

int err[4];

int indice;

string encontrado;

endvar;

indice = 0;

while(indice < 4)

{

err[indice] = indice;

indice = indice + 1;

}

indice = 0;

while(indice < 4)

{

if(err[indice] == n)

{

encontrado = "Encontrado";

}

else

{

encontrado = "No encontrado";

}

indice = indice + 1;

}

print(encontrado);

}

main ( ) {

wow.holis();

a = 5;

a = fibonacci(5);;

print(a);

a = 5;

wow.x = 3;

wii.x = wow.x;

print(a);

arr[a-3] = 2;

dosie[a-4][arr[a-3]] = 4;

print(dosie[a-4][arr[a-3]]);

d = 3.4;

b = 2 \* a;

print(b);

print(arr[a-3]);

arr2[1+2] = 4.5;

print(arr2[1+2]);

print(d);

a = 1 + 2 + 3 + 4 + 5 + 6 + 7;

a = 5 - 2 + 1;

print(a);

str = "hola";

boolean = true;

a = 2 + 4;

busqueda(3);

print(a);

}

}

* 1. Testcase 2

func int factorial(int n)

{

int sum;

int inicio;

endvar;

sum = 1;

inicio = 1;

if (n <= 1)

{

return(sum);

}

while (n > 1)

{

sum = sum \* n;

n = n - 1;

}

return(sum);

}

1. Listado del código
   1. Gramática sin reglas

################################################################################

'''

Creación de la estructura de Tabla de Variables

'''

var = collections.namedtuple('Var', 'type dim dirV')

varname = ''

vartype = ''

vardim1 = ''

vardim2 = ''

m0 = 0

pDim = collections.namedtuple('PDim', 'id dim')

pilaDim = []

retOp = ''

retTyp = ''

################################################################################

Dim = collections.namedtuple('Dim', 'lsup linf mn')

dim = {}

################################################################################

'''

Creación de la estructura de Directorio de Procedimientos

'''

DirProc = collections.namedtuple('DirProc', 'tipoRet params vars inicio')

dirProc = {}

modRet = ''

modActual = ''

modSiguiente = ''

modParamType = ''

modParams = 1

retornoProc = {}

retPr = collections.namedtuple('RetPr', 'dirV type')

################################################################################

################################################################################

'''

Creación de la estructura para los objetos

'''

Objeto = collections.namedtuple('Objeto', 'herencia vars mods')

objetos = {}

#Variable de objeto, tiene visibilidad, tipo y dimensión

VarObj = collections.namedtuple('VarObj', 'vis type dim')

varobjs = {}

objActual = ''

visibilidad = ''

################################################################################

################################################################################

'''

Creación de la lista de cuádruplos

'''

Cuadruplo = collections.namedtuple('Cuadruplo', ' pos op oper1 oper2 res')

cuad = []

cont = 0

################################################################################

#Especificación de precedencia de operadores

precedence = (

('nonassoc', 'MENQUE','MAYQUE','MAYIGU','MENIGU','DIFERE'),

('left', 'AND', 'OR'),

('left', 'MAS', 'MENOS'),

('left', 'POR', 'DIVISI'),

('right', 'UMENOS', 'IGUAL') #UMENOS es el negativo UNARIO

)

#REGLAS DEL LENGUAJE

################################################################################

# PROGRAM

def p\_program(p):

'program : PROGRAM IDENTI semCreaDP cuadInicial LBRACK progobj vars progmod main RBRACK'

print("\n\*\*\*Entrada aceptada\*\*\*\n")

def p\_progobj(p):

'''progobj : objeto progobj

|'''

def p\_progmod(p):

'''progmod : modulo progmod

|'''

#####SEMÁNTICA DE PROGRAM

def p\_semCreaDP(p):

'semCreaDP : '

def p\_cuadInicial(p):

'cuadInicial :'

#MAIN

def p\_main(p):

'main : MAIN LPAREN RPAREN iniciaMain bloque'

def p\_iniciaMain(p):

'iniciaMain :'

#OBJETO

def p\_objeto(p):

'objeto : OBJECT IDENTI foundObjID herencia LBRACK objvars vieneAsignacion objmet RBRACK acabaOBJ'

def p\_herencia(p):

'''herencia : LPAREN IDENTI RPAREN

|'''

def p\_objmet(p):

'''objmet : modulo objmet

|'''

def p\_foundObjID(p):

'foundObjID :'

def p\_vieneAsignacion(p):

'''vieneAsignacion : asignacion masAsign

| '''

def p\_masAsign(p):

'''masAsign : asignacion masAsign

| '''

def p\_acabaOBJ(p):

'acabaOBJ :'

#BLOQUE

def p\_bloque(p):

'bloque : LBRACK bloq RBRACK'

def p\_bloq(p):

'''bloq : estret

|'''

def p\_estret(p):

'''estret : estatuto bloq

| return bloq

| varcte bloq'''

#ESTATUTO

def p\_estatuto(p):

'''estatuto : lectura

| asignacion

| condicion

| escritura

| ciclo'''

#Ver sobre la regla de METCALL

#VARS

def p\_vars(p):

'''vars : tipo IDENTI vartam SEMCOL insertVar varsagain'''

def p\_vartam(p):

'''vartam : tamano

|'''

def p\_varsagain(p):

'''varsagain : vars

| ENDVAR SEMCOL'''

def p\_objvars(p):

'objvars : visibilidad tipo IDENTI vartam insertObjVar SEMCOL delVis objvarsagain'

def p\_visibilidad(p):

'''visibilidad : PUBLIC foundKeyword

| PRIVATE foundKeyword'''

def p\_foundKeyword(p):

'foundKeyword :'

def p\_insertObjVar(p):

'insertObjVar :'

def p\_delVis(p):

'delVis :'

def p\_objvarsagain(p):

'''objvarsagain : objvars

| ENDVAR SEMCOL'''

def p\_insertVar(p):

'insertVar : '

#TAMAÑO

def p\_tamano(p):

'tamano : LCORCH CONINT metearr RCORCH tammatriz'

global vardim1

def p\_metearr(p):

'metearr :'

#TAMMATRIZ

def p\_tammatriz(p):

'''tammatriz : LCORCH CONINT RCORCH

| '''

#RETURN

def p\_return(p):

'return : RETURN superexpresion SEMCOL'

#ASIGNACION

def p\_asignacion(p):

'asignacion : IDENTI tieneObjeto IGUAL foundEQ exp SEMCOL'

def p\_tieneObjeto(p):

'''tieneObjeto : PUNTO IDENTI foundID

| foundID

| dimension '''

def p\_foundID(p):

'foundID :'

def p\_foundEQ(p):

'foundEQ :'

#ESCRITURA

def p\_escritura(p):

'escritura : PRINT LPAREN prntparam RPAREN SEMCOL'

def p\_prntparam(p):

'''prntparam : superexpresion

|'''

#LECTURA

def p\_lectura(p):

'lectura : READ LPAREN IDENTI dimobj RPAREN SEMCOL'

def p\_dimobj(p):

'''dimobj : dimension

| PUNTO IDENTI'''

#CONDICION - IF

def p\_condicion(p):

'condicion : IF LPAREN superexpresion RPAREN genCuadIF bloque tieneelse'

def p\_tieneelse(p):

'''tieneelse : ELSE genCuadElse bloque acabaIF

| acabaIF'''

def p\_genCuadIF(p):

'genCuadIF :'

def p\_acabaIF(p):

'acabaIF :'

def p\_genCuadElse(p):

'genCuadElse :'

#CICLO

def p\_ciclo(p):

'''ciclo : dowhile

| for

| while'''

#DOWHILE

def p\_dowhile(p):

'dowhile : DO doMeteCont bloque WHILE LPAREN superexpresion RPAREN SEMCOL doTermina'

def p\_doMeteCont(p):

'doMeteCont :'

def p\_doTermina(p):

'doTermina :'

#WHILE

def p\_while(p):

'while : WHILE meteContWhile LPAREN superexpresion RPAREN whileGeneraFalso bloque whileTermina'

def p\_meteContWhile(p):

'meteContWhile :'

def p\_whileGeneraFalso(p):

'whileGeneraFalso :'

def p\_whileTermina(p):

'whileTermina :'

#FOR

def p\_for(p):

'for : FOR LPAREN asignacion meteContWhile superexpresion SEMCOL whileGeneraFalso asignacion RPAREN bloque whileTermina'

#DIMENSION

def p\_dimension(p):

'''dimension : LCORCH esDimensionada exp RCORCH segParte matriz'''

#MATRIZ

def p\_matriz(p):

'''matriz : LCORCH exp sonDosDims RCORCH

| soloUnaDim'''

def p\_esDimensionada(p):

'esDimensionada : '

def p\_segParte(p):

'segParte : '

def p\_soloUnaDim(p):

'soloUnaDim : '

def p\_sonDosDims(p):

'sonDosDims : '

#TERMINO

def p\_termino(p):

'termino : factor term'

def p\_term(p):

'''term : POR foundMUDI factor genCuadMUDI term

| DIVISI foundMUDI factor genCuadMUDI term

|'''

def p\_foundMUDI(p):

'foundMUDI :'

def p\_genCuadMUDI(p):

'genCuadMUDI :'

#FACTOR

def p\_factor(p):

'''factor : LPAREN foundLP exp RPAREN foundRP

| menunario varcte'''

def p\_menunario(p):

'''menunario : MENOS %prec UMENOS

|'''

def p\_foundLP(p):

'foundLP :'

pOper.append(p[-1])

def p\_foundRP(p):

'foundRP :'

#MODULO

def p\_modulo(p):

'modulo : FUNC modtipo IDENTI semCreaMod LPAREN modparams RPAREN modbloque'

def p\_modtipo(p):

'''modtipo : tipos

| VOID'''

def p\_modparams(p):

'''modparams : tipos IDENTI foundParam masmodparams

|'''

def p\_foundParam(p):

'foundParam :'

def p\_masmodparams(p):

'''masmodparams : SEMCOL modparams

|'''

def p\_modbloque(p):

'modbloque : LBRACK noVars RBRACK retornoMOD'

def p\_noVars(p):

'''noVars : meteInicio vars modbloq

| meteInicio modbloq'''

def p\_retornoMOD(p):

'retornoMOD : '

def p\_meteInicio(p):

'meteInicio : '

def p\_modbloq(p):

'''modbloq : estatuto modbloq

| return modbloq

| varcte modbloq

|'''

#####SEMÁNTICA DE MODULO

def p\_semCreaMod(p):

'semCreaMod :'

#VARCTE

def p\_varcte(p):

'''varcte : otrasconst

| cteid'''

def p\_cteid(p):

'cteid : IDENTI funcdimobj'

def p\_funcdimobj(p):

'''funcdimobj : metodoCall

| dimension

| PUNTO IDENTI llamaObjeto limpia

| foundID''' #dimobjcall

def p\_limpia(p):

'limpia : '

def p\_metodoCall(p):

'metodoCall : LPAREN generaERA llamaParam RPAREN resetPARAM SEMCOL'

global objActual

def p\_llamaObjeto(p):

'''llamaObjeto : foundID

| metodoCall

| dimension'''

def p\_llamaParam(p):

'''llamaParam : exp generaPARAM mascall

|'''

def p\_mascall(p):

'''mascall : COMMA exp generaPARAM mascall

|'''

def p\_generaERA(p):

'generaERA : '

def p\_generaPARAM(p):

'generaPARAM : '

def p\_resetPARAM(p):

'resetPARAM : '

def p\_otrasconst(p):

'''otrasconst : CONFLO foundNUMFLO

| CONINT foundNUMINT

| conbol

| CONSTR foundSTR'''

def p\_foundNUMINT(p):

'foundNUMINT :'

def p\_foundNUMFLO(p):

'foundNUMFLO :'

def p\_foundSTR(p):

'foundSTR :'

def p\_conbol(p):

'''conbol : TRUE foundBOOL

| FALSE foundBOOL'''

def p\_foundBOOL(p):

'foundBOOL :'

#TIPO

def p\_tipo(p):

'''tipo : tipos

| IDENTI'''

'''def p\_isobject(p):

'isobject : '

global vartype

if p[-1] in objetos:

vartype = p[-1]

else:

sys.exit '''

#TIPOS

def p\_tipos(p):

'''tipos : INT

| FLOAT

| STRING

| BOOL'''

#EXPRESION

def p\_expresion(p):

'''expresion : exp compara'''

def p\_compara(p):

'''compara : MENQUE foundComp exp genCuadMUDI

| MAYQUE foundComp exp genCuadMUDI

| DIFERE foundComp exp genCuadMUDI

| MENIGU foundComp exp genCuadMUDI

| MAYIGU foundComp exp genCuadMUDI

| IGUIGU foundComp exp genCuadMUDI

|'''

def p\_foundComp(p):

'foundComp :'

#EXP

def p\_exp(p):

'exp : termino expre'

def p\_expre(p):

'''expre : MAS foundPLMI termino genCuadMUDI expre

| MENOS foundPLMI termino genCuadMUDI expre

|'''

def p\_foundPLMI(p):

'foundPLMI :'

#SUPEREXPRESION

def p\_superexpresion(p):

'superexpresion : expresion superexp'

def p\_superexp(p):

'''superexp : logicos

|'''

def p\_logicos(p):

'logicos : andor expresion genCuadMUDI maslogicos'

def p\_andor(p):

'''andor : AND foundANDOR

| OR foundANDOR'''

def p\_foundANDOR(p):

'foundANDOR :'

def p\_maslogicos(p):

'''maslogicos : logicos

|'''

#ERRORES DE PARSING

def p\_error(p):

sys.exit("Error de Sintaxis en '%s', renglón '%s'" % (p.value, p.lexer.lineno))

* 1. Funciones de la Máquina virtual

##### SUMA

if op == 0:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if type(operSuma1) is float or type(operSuma2) is float:

MemTemporal.float[res] = operSuma1 + operSuma2

else:

MemTemporal.int[res] = operSuma1 + operSuma2

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

######### RESTA

elif op == 1:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if type(operSuma1) is float or type(operSuma2) is float:

MemTemporal.float[res] = operSuma1 - operSuma2

else:

MemTemporal.int[res] = operSuma1 - operSuma2

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

########### PRODUCTO

elif op == 2:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if type(operSuma1) is float or type(operSuma2) is float:

MemTemporal.float[res] = operSuma1 \* operSuma2

else:

MemTemporal.int[res] = operSuma1 \* operSuma2

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 3:

########### DIVISION

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if type(operSuma1) is float or type(operSuma2) is float:

MemTemporal.float[res] = operSuma1 / operSuma2

else:

MemTemporal.int[res] = operSuma1 / operSuma2

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 4:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 < operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 5:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 > operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 6:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 != operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 7:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 <= operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 8:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 >= operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 9:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 == operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 10:

if type(res) is str:

dirDest = int(re.search(r'\d+', res).group())

if 22500 <= dirDest < 25000:

#print(MemLocal.int)

destino = MemTemporal.int[dirDest]

#print(destino, oper1)

if 2500 <= destino < 5000:

if 2500 <= oper1 < 5000:

MemGlobal.int[destino] = MemGlobal.int[oper1]

elif 12500 <= oper1 < 15000:

MemGlobal.int[destino] = MemLocal.int[oper1]

elif 32500 <= oper1 < 35000:

MemGlobal.int[destino] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

MemGlobal.int[destino] = MemTemporal.int[oper1]

elif 5000 <= destino < 7500:

if 5000 <= oper1 < 7500:

MemGlobal.float[destino] = MemGlobal.float[oper1]

elif 15000 <= oper1 < 17500:

MemGlobal.float[destino] = MemLocal.float[oper1]

elif 25000 <= oper1 < 27500:

MemGlobal.float[destino] = MemTemporal.float[oper1]

elif 35000 <= oper1 < 37500:

MemGlobal.float[destino] = invMemConstFLO[oper1]

elif 12500 <= destino < 15000:

if 2500 <= oper1 < 5000:

MemLocal.int[destino] = MemGlobal.int[oper1]

elif 12500 <= oper1 < 15000:

MemLocal.int[destino] = MemLocal.int[oper1]

elif 32500 <= oper1 < 35000:

MemLocal.int[destino] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

MemLocal.int[destino] = MemTemporal.int[oper1]

#print(MemTemporal.int)

#Es una variable global entera

elif 2500 <= res < 5000:

#if res in MemGlobal.int:

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

destino = MemTemporal.int[dirDest]

oper1 = destino

if 2500 <= oper1 < 5000:

MemGlobal.int[res] = MemGlobal.int[oper1]

elif 12500 <= oper1 < 15000:

MemGlobal.int[res] = MemLocal.int[oper1]

elif 32500 <= oper1 < 35000:

MemGlobal.int[res] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

MemGlobal.int[res] = MemTemporal.int[oper1]

'''else:

if 2500 <= oper1 < 5000:

MemGlobal.int[res] = MemGlobal.int[oper1]

elif 32500 <= oper1 < 35000:

MemGlobal.int[res] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

MemGlobal.int[res] = MemTemporal.int[oper1]'''

#Es una variable global Flotante

elif 5000 <= res < 7500:

#if res in MemGlobal.float:

if 2500 <= oper1 < 5000:

MemGlobal.float[res] = MemGlobal.int[oper1]

elif 12500 <= oper1 < 15000:

MemGlobal.float[res] = MemLocal.int[oper1]

elif 32500 <= oper1 < 35000:

MemGlobal.float[res] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

MemGlobal.float[res] = MemTemporal.int[oper1]

elif 5000 <= oper1 < 7500:

MemGlobal.float[res] = MemGlobal.float[oper1]

elif 15000 <= oper1 < 17500:

MemGlobal.float[res] = MemLocal.float[oper1]

elif 25000 <= oper1 < 27500:

MemGlobal.float[res] = MemTemporal.float[oper1]

elif 35000 <= oper1 < 37500:

MemGlobal.float[res] = invMemConstFLO[oper1]

#Es una Variable global Booleana

elif 7500 <= res < 10000:

pass

#Es una variable global string

elif 10000 <= res < 12500:

if 40000 <= oper1 < 42500:

MemGlobal.str[res] = invMemConstSTR[oper1]

#LOCALES

elif 12500 <= res < 15000:

if 32500 <= oper1 < 35000:

stackLocal[len(stackLocal) - 1].int[res] = invMemConstINT[oper1]

MemLocal.int[res] = invMemConstINT[oper1]

elif 22500 <= oper1 < 25000:

stackLocal[len(stackLocal) - 1].int[res] = MemTemporal.int[oper1]

MemLocal.int[res] = MemTemporal.int[oper1]

elif 12500 <= oper1 < 25000:

stackLocal[len(stackLocal) - 1].int[res] = stackLocal[len(stackLocal) - 1].int[oper1]

MemLocal.int[res] = stackLocal[len(stackLocal) - 1].int[oper1]

if pos < len(cuad) - 1:

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

else:

print("DONE")

elif op == 11:

operSuma1 = operSuma2 = 0

#print(oper1, oper2, res)

#OPERADOR 1

#Arreglos para globales y locales

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma1 = MemLocal.int[destino]

########## ENTEROS

#Constantes

elif 32500 <= oper1 < 35000:

operSuma1 = invMemConstINT[oper1]

elif 35000 <= oper1 < 37500:

operSuma1 = invMemConstFLO[oper1]

#Temporales

elif 22500 <= oper1 < 25000:

operSuma1 = MemTemporal.int[oper1]

elif 25000 <= oper1 < 27500:

operSuma1 = MemTemporal.float[oper1]

#Globales

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int:

operSuma1 = MemGlobal.int[oper1]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operSuma1 = MemGlobal.float[oper1]

else:

sys.exit("Variable en suma NO inicializada")

#Locales

elif 12500 <= oper1 < 15000:

operSuma1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 15000 <= oper1 < 17500:

operSuma1 = stackLocal[len(stackLocal) - 1].float[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operSuma2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operSuma2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operSuma2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operSuma2 = MemLocal.int[destino]

elif 32500 <= oper2 < 35000:

operSuma2 = invMemConstINT[oper2]

elif 35000 <= oper2 < 37500:

operSuma2 = invMemConstFLO[oper2]

elif 22500 <= oper2 < 25000:

operSuma2 = MemTemporal.int[oper2]

elif 25000 <= oper2 < 27500:

operSuma2 = Memtemporal.float[oper2]

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int:

operSuma2 = MemGlobal.int[oper2]

else:

sys.exit("Variable en suma NO inicializada")

elif 5000 <= oper2 < 7500:

if oper2 in MemGlobal.float:

operSuma2 = MemGlobal.float[oper2]

else:

sys.exit("Variable en suma no inicializada")

elif 12500 <= oper2 < 15000:

operSuma2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 15000 <= oper2 < 17500:

operSuma2 = stackLocal[len(stackLocal) - 1].float[oper2]

#print(oper2)

if operSuma1 and operSuma2:

MemTemporal.bool[res] = True

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 12:

#OPERADOR 1

operador1 = operador2 = 0

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operador1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operador1 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operador1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operador1 = MemLocal.int[destino]

#Es entero

elif 2500 <= oper1 < 5000:

if oper1 in MemGlobal.int[oper1]:

operador1 = MemGlobal.int[oper1]

else:

sys.exit("Variable no inicializada")

elif 12500 <= oper1 < 15000:

operador1 = stackLocal[len(stackLocal) - 1].int[oper1]

elif 22500 <= oper1 < 25000:

operador1 = MemTemporal.int[oper1]

elif 32500 <= oper1 < 35000:

operador1 = invMemConstINT[oper1]

#Es flotante

elif 5000 <= oper1 < 7500:

if oper1 in MemGlobal.float:

operador1 = MemGlobal.float[oper1]

else:

sys.exit("Variable no inicializada")

elif 15000 <= oper1 < 17500:

operador1 = stackLocal[len(stackLocal) - 1].float[oper1]

elif 25000 <= oper1 < 27500:

operador1 = MemTemporal.float[oper1]

elif 35000 <= oper1 < 37500:

operador1 = invMemConstFLO[oper1]

#Es boolean

elif 7500 <= oper1 < 10000:

if oper1 in MemGlobal.bool:

operador1 = MemGlobal.bool[oper1]

else:

sys.exit("Variable no inicializada")

elif 17500 <= oper1 < 20000:

operador1 = stackLocal[len(stackLocal) -1].bool[oper1]

elif 27500 <= oper1 < 30000:

operador1 = MemTemporal.bool[oper1]

elif 37500 <= oper1 < 40000:

operador1 = invMemConstBOL[oper1]

#OPERADOR 2

if type(oper2) is str:

dirDest = int(re.search(r'\d+', oper2).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

operador2 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

operador2 = MemGlobal.float[destino]

elif 12500 <= destino < 15000:

operador2 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

operador2 = MemLocal.int[destino]

#Es Entero

elif 2500 <= oper2 < 5000:

if oper2 in MemGlobal.int[oper2]:

operador2 = MemGlobal.int[oper2]

else:

sys.exit("Varibale no inicializada")

elif 12500 <= oper2 < 15000:

operador2 = stackLocal[len(stackLocal) - 1].int[oper2]

elif 22500 <= oper2 < 25000:

pass

elif 32500 <= oper2 < 35000:

operador2 = invMemConstINT[oper2]

#Es flotante

elif 5000 <= oper2 < 7500:

if oper1 in MemGlobal.float:

operador2 = MemGlobal.float[oper2]

else:

sys.exit("Variable no inicializada")

elif 15000 <= oper2 < 17500:

operador2 = stackLocal[len(stackLocal) - 1].float[oper2]

elif 25000 <= oper2 < 27500:

operador2 = MemTemporal.float[oper2]

elif 35000 <= oper2 < 37500:

operador2 = invMemConstFLO[oper2]

#Es boolean

elif 7500 <= oper2 < 10000:

if oper2 in MemGlobal.bool:

operador2 = MemGlobal.bool[oper2]

else:

sys.exit("Variable no inicializada")

elif 17500 <= oper2 < 20000:

operador2 = stackLocal[len(stackLocal) -1].bool[oper2]

elif 27500 <= oper2 < 30000:

operador2 = MemTemporal.bool[oper2]

elif 37500 <= oper2 < 40000:

operador2 = invMemConstBOL[oper2]

if operador1 or operador2:

MemTemporal.bool[res] = True

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

else:

MemTemporal.bool[res] = False

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 14: #PRINT

if type(res) is str:

dirDest = int(re.search(r'\d+', res).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int[res])

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

res = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

res = MemGlobal.float[destino]

elif 7500 <= destino < 10000:

res = MemGlobal.bool[destino]

elif 10000 <= destino < 12500:

res = MemGlobal.str[destino]

elif 12500 <= destino < 15000:

res = MemLocal.int[destino]

elif 15000 <= destino < 17500:

res = MemLocal.float[destino]

elif 17500 <= destino < 20000:

res = MemLocal.bool[destino]

elif 20000 <= destino < 22500:

res = MemLocal.str[destino]

elif 22500 <= destino < 25000:

res = MemTemporal.int[destino]

elif 25000 <= destino < 27500:

res = MemTemporal.float[destino]

elif 27500 <= destino < 30000:

res = MemTemporal.bool[destino]

elif 30000 <= desitno < 32500:

res = MemTemporal.str[detino]

#GLOBALES

if 2500 <= res < 5000:

print(MemGlobal.int[res])

elif 5000 <= res < 7500:

print(MemGlobal.float[res])

elif 7500 <= res < 10000:

print(MemGlobal.bool[res])

elif 10000 <= res < 12500:

print(MemGlobal.str[res])

#LOCALES

elif 12500 <= res < 15000:

print(MemLocal.int[res])

elif 15000 <= res < 17500:

print(MemLocal.float[res])

elif 17500 <= res < 20000:

print(MemLocal.bool[res])

elif 20000 <= res < 22500:

print(MemLocal.str[res])

#CONSTANTES

elif 32500 <= res < 35000:

print(invMemConstINT[res])

elif 35000 <= res < 37500:

print(invMemConstFLO[res])

elif 37500 <= res < 40000:

print(invMemConstBOL[res])

elif 40000 <= res < 42500:

print(invMemConstSTR[res])

#Temporales

elif 22500 <= res < 25000:

print(MemTemporal.int[res])

elif 25000<= res < 275000:

print(MemTemporal.float[res])

elif 27500 <= res < 30000:

print(MemTemporal.bool[res])

elif 30000 <= res < 32500:

print(MemTemporal.str[res])

if pos < (len(cuad) - 1):

MaquinaVirtual(cuad[(pos+1)].pos, cuad[(pos+1)].op, cuad[(pos+1)].oper1, cuad[(pos+1)].oper2, cuad[(pos+1)].res)

else:

print("DONE")

elif op == 15:

#print("Hola RETURN")

MaquinaVirtual(cuad[(pos+1)].pos, cuad[(pos+1)].op, cuad[(pos+1)].oper1, cuad[(pos+1)].oper2, cuad[(pos+1)].res)

elif op == 16: #GOTOF

if MemTemporal.bool[oper1] == False:

MaquinaVirtual(cuad[res].pos, cuad[res].op, cuad[res].oper1, cuad[res].oper2, cuad[res].res)

elif MemTemporal.bool[oper1] == True:

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 17:

print("Hay GOTOV")

if MemTemporal.bool[oper1] == True:

MaquinaVirtual(cuad[res].pos, cuad[res].op, cuad[res].oper1, cuad[res].oper2, cuad[res].res)

elif MemTemporal.bool[oper1] == False:

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 18:

MaquinaVirtual(cuad[res].pos, cuad[res].op, cuad[res].oper1, cuad[res].oper2, cuad[res].res)

elif op == 19:

stackLocal.append(MemLocal)

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 20:

stackEstado.append(pos + 1)

MaquinaVirtual(cuad[res].pos, cuad[res].op, cuad[res].oper1, cuad[res].oper2, cuad[res].res)

elif op == 21:

#PARAMETROS

global MemLocalINTCont, MemLocalINTOffset, MemLocalFLOCont, MemLocalFLOOffset, MemLocalBOLCont, MemLocalBOLOffset

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

if 2500 <= destino < 5000:

oper1 = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

oper1 = MemGlobal.float[destino]

elif 7500 <= destino < 10000:

oper1 = MemGlobal.bool[destino]

elif 10000 <= destino < 12500:

oper1 = MemGlobal.str[destino]

elif 12500 <= destino < 15000:

oper1 = MemLocal.int[destino]

elif 15000 <= destino < 17500:

oper1 = MemLocal.int[destino]

elif 17500 <= destino < 20000:

oper1 = MemLocal.bool[destino]

elif 20000 <= destino < 25000:

oper1 = MemLocal.str[destino]

if 2500 <= oper1 < 5000:

parametro = MemGlobal.int[oper1]

stackLocal[len(stackLocal) - 1].int[MemLocalINTCont + MemLocalINTOffset] = parametro

MemLocalINTOffset += 1

#print(stackLocal)

elif 5000 <= oper1 < 7500:

parametrp = MemGlobal.float[oper1]

stackLocal[len(stackLocal) - 1].int[MemLocalFLOCont + MemLocalFLOOffset]

elif 12500 <= oper1 < 15000:

parametro = stackLocal[len(stackLocal)-2].int[oper1]

stackLocal[len(stackLocal)-1].int[MemLocalINTCont + MemLocalINTOffset] = parametro

MemLocalINTOffset += 1

elif 22500 <= oper1 < 25000:

parametro = MemTemporal.int[oper1]

stackLocal[len(stackLocal) - 1].int[MemLocalINTCont + MemLocalINTOffset] = parametro

MemLocalINTOffset += 1

elif 32500 <= oper1 < 35000:

parametro = invMemConstINT[oper1]

#print(MemLocal)

stackLocal[len(stackLocal)-1].int[MemLocalINTCont + MemLocalINTOffset] = parametro

MemLocal.int[MemLocalINTCont + MemLocalINTOffset] = parametro

MemLocalINTOffset += 1

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

elif op == 22:

siguienteCuad = stackEstado.pop()

stackLocal.pop()

if siguienteCuad < (len(cuad) - 1):

MaquinaVirtual(cuad[siguienteCuad].pos, cuad[siguienteCuad].op, cuad[siguienteCuad].oper1, cuad[siguienteCuad].oper2, cuad[siguienteCuad].res)

else:

print("DONE")

elif op == 23:

print("Hay READ")

elif op == 24:

#print(MemGlobal.int)

#print(oper1, oper2, res, invMemConstINT, MemTemporal)

if type(oper1) is str:

dirDest = int(re.search(r'\d+', oper1).group())

if 22500 <= dirDest < 25000:

#print(MemTemporal.int)

destino = MemTemporal.int[dirDest]

#print(MemGlobal.int)

#print(oper1, oper2, res)

if 2500 <= destino < 5000:

indice = MemGlobal.int[destino]

elif 5000 <= destino < 7500:

if 5000 <= oper1 < 7500:

MemGlobal.float[destino] = MemGlobal.float[oper1]

elif 15000 <= oper1 < 17500:

MemGlobal.float[destino] = MemLocal.float[oper1]

elif 25000 <= oper1 < 27500:

MemGlobal.float[destino] = MemTemporal.float[oper1]

elif 35000 <= oper1 < 37500:

MemGlobal.float[destino] = invMemConstFLO[oper1]

elif 22500 <= oper1 < 25000:

indice = MemTemporal.int[oper1]

if indice < oper2 or indice > res:

sys.exit("Índice fuera de rango %s" % inidice)

else:

MaquinaVirtual(cuad[pos+1].pos, cuad[pos+1].op, cuad[pos+1].oper1, cuad[pos+1].oper2, cuad[pos+1].res)

Manual de Usuario

Marfle es un lenguaje orientado a objetos. La estructura de los programas es similar a Java, aunque con algunas diferencias al momento de construir el programa en cuestión.

Un programa siempre debe comenzar con la palabra ‘program’, seguido del código que estará encapsulado dentro de dos llaves. Marfle cuenta con objetos (clases), creación de métodos, y el main.

main() { … }

func <tipo> <nombre> (<tipo> <argumento>) { … }

objeto <nombre> { … }

Los métodos usan ‘func’ antes del nombre, y los objetos ‘objeto’. El main se declara tal como es.

Las variables, como en java, se les debe declarar el tipo al momento de nombrarlas. Los ciclos también hacen uso de las mismas reglas.

Para un ejemplo de cómo escribir un programa, vea la referencia en las Pruebas de Funcionamiento.