

**Noobscript**

Diseño de Compiladores

Dr. José Icaza / Ing. Elda Quiroga

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Víctor Manuel Clemente López Jorge Antonio Carrillo Curiel

A00800053 A0113698

Monterrey Nuevo León, a 6 de mayo 2015

Table of Contents

Descripción y documentación del proyecto 3

Visión 3

Objetivo del lenguaje y alcance 3

Análisis de requerimientos y casos de uso 3

Proceso General 7

Descripción del lenguaje 9

Nombre de el Lenguaje 9

Descripción genérica de características del lenguaje 9

Posibles errores 9

Descripción del compilador 10

Hardware y Software 10

Descripción del análisis léxico 10

Descripción del análisis de sintaxis 12

Descripción del análisis semántico 16

Descripción de la máquina virtual 22

Equipo de computo 22

Descripción de el uso de memoria 22

Listado con los principales módulos de el compilador 23

Bibliografía

Noobscript Manual de usuario rápido

# Descripción y documentación del proyecto

## Visión

Definir como objeto de estudio la creación del proyecto de programación, consistente en crear un nuevo lenguaje de programación orientado a objetos el cual considera todos los aspectos y elementos básicos de un lenguaje de una manera más sencilla para que, de esta manera, se agilice el entendimiento de los mismos y dar una visión más clara a el estudio abstracto del desarrollo computacional.

## Objetivo del lenguaje y alcance

El estudio y desarrollo de lenguajes de programación es un estudio abstracto y complicado desde el punto de vista de lógica, ya que al contener elementos como ciclos o condiciones su aprendizaje puede llegar a ser altamente complicado. Se pretende diseñar un lenguaje de programación básico orientado objetos que permita la definición de clases con atributos y métodos de una manera más sencilla, para de este modo, agilizar el proceso de aprendizaje de los mismos en un ambiente de programación más amistoso orientado a la educación.

## Análisis de requerimientos y casos de uso

Se analiza como caso de estudio los diferentes escenarios en el que será usado el lenguaje de programación, en el cual puede ser accedido por un desarrollador en caso de querer crear o modificar un programa y por un usuario que hace uso de el programa desarrollado, el desarrollador también puede hacer testing de los programas creados.

Casos de uso:

1. Usar Programa

2. Testing Programa

3. Crear programa

4. Interpretar Programa

5. Compilar Programa

6. Editar Programa

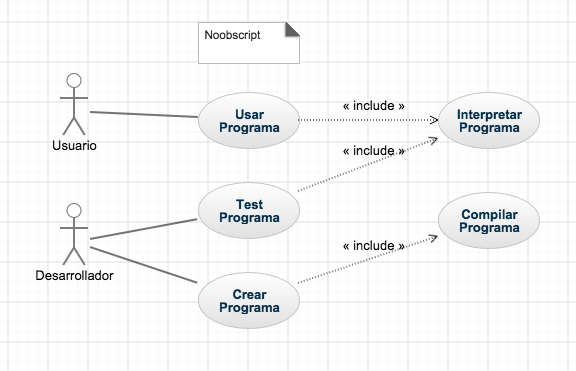
|  |  |  |
| --- | --- | --- |
| **ID** | EJ-01 | |
| **Nombre** | Usar Progama | |
| **Autores** | Victor Clemente | |
| **Descripción** | El usuario o desarrollador ejecuta el programa echo por el desarrollador en su ordenador. | |
| **Precondición** | Ninguna | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario/desarrollador inicializa la ejecución de el programa desarrollado. |
| 2 | El usuario termina la ejecución de el programa y este de cierra. |
| **Flujo alternativo** | **Paso** | **Acción** |
| a.1 | El programa tiene errores de ejecución y se cierra. |
| **Postcondiciones** | Ninguna | |
| **Comentarios** | Ninguno | |

|  |  |  |
| --- | --- | --- |
| **ID** | TE-02 | |
| **Nombre** | Testing programa | |
| **Autores** | Jorge Curiel | |
| **Descripción** | El desarrollador hace pruebas de el programa creado para encontrar errores. | |
| **Precondición** | Ninguna | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El desarrollador inicia la ejecución de el programa en el ordenador |
| 2 | El desarrollador realiza las pruebas que sean necesarias para comprobar el funcionamiento de el programa. |
| 3 | El desarrollador termina la ejecución de el programa y este de cierra. |
| **Flujo alternativo** | **Paso** | **Acción** |
| a.1  a.2 | El programa tiene errores de ejecución y se cierra. |
| **Postcondiciones** | Ninguna | |
| **Comentarios** | Ninguno | |

|  |  |  |
| --- | --- | --- |
| **ID** | CR-03 | |
| **Nombre** | Crear Programa | |
| **Autores** | Victor Clemente | |
| **Descripción** | El desarrollador realiza el código de el programa el cual estará listo para su compilación o interpretación. | |
| **Precondición** | Ninguna | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El desarrollador elabora el código noobscript |
| **Flujo alternativo** | **Paso** | **Acción** |
| a.1 | Ninguno |
| **Postcondiciones** | Se tiene código listo para interpretarse o compilarse. | |
| **Comentarios** | Ninguno | |

|  |  |  |
| --- | --- | --- |
| **ID** | EJ-04 | |
| **Nombre** | Intepretar Programa | |
| **Autores** | Victor Clemente | |
| **Descripción** | El ordenador interpreta el código de noobscipt el cual ya fue compilado y es ejecutado en el compilador respecto a las instrucciones asignadas. | |
| **Precondición** | Código noobscript compilado | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El ordenador recibe el código ya interpretado |
| 2 | El ordenador realiza la ejecución necesaria respecto al código recibido |
| 3 | El programa termina de interpretarse y avisa que ha terminado, |
| **Flujo alternativo** | **Paso** | **Acción** |
| a.1  a.2 | El programa tiene errores de ejecución y se cierra. |
| **Postcondiciones** | Ninguna | |
| **Comentarios** | Ninguno | |

|  |  |  |
| --- | --- | --- |
| **ID** | CO-05 | |
| **Nombre** | Compilar Programa | |
| **Autores** | Jorge Curiel | |
| **Descripción** | El compilador recibe el código echo por el desarrollador y lo traduce para que pueda ser ejecutado por el ordenador. | |
| **Precondición** | Código noobscript | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El compilador recibe el código de el desarrollador. |
| 2 | El compilador realiza el análisis léxico, sintáctico y semántico para generar código maquina para que sea interpretado. |
| 3 | El compilador termina con el proceso de compilación y deja de ejecutarse. |
| **Flujo alternativo** | **Paso** | **Acción** |
| a.1  a.2 | El programa tiene errores de algún tipo, avisa que tipo de error es y se deja de ejecutar el proceso de compilación. |
| **Postcondiciones** | Genera código intermedio. | |
| **Comentarios** | Ninguno | |



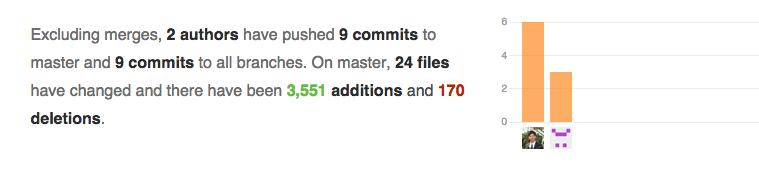
## Proceso General

El desarrollo en sí no comenzó hasta el día 19 de abril, ya que durante el tiempo pasado se dedicó al análisis y al desarrollo de la arquitectura en base a los requisitos no funcionales de nuestro lenguaje.

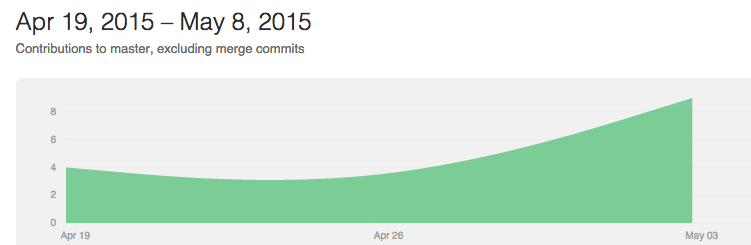
Durante el proceso de preparación se desarrollo un análisis léxico, sintáctico y semántico para poder definir abstractamente las características de el lenguaje y en base a ellas poder desarrollarse.

Ya en el desarrollo, se utilizó la plataforma github para llevar a cabo control de versiones y de avances, mismo repositorio que permite conseguir un desempeño más óptimo gracias a las facilidades que este ofrece.

Se muestran a continuación las gráficas y bitácoras que se generan del proyecto.



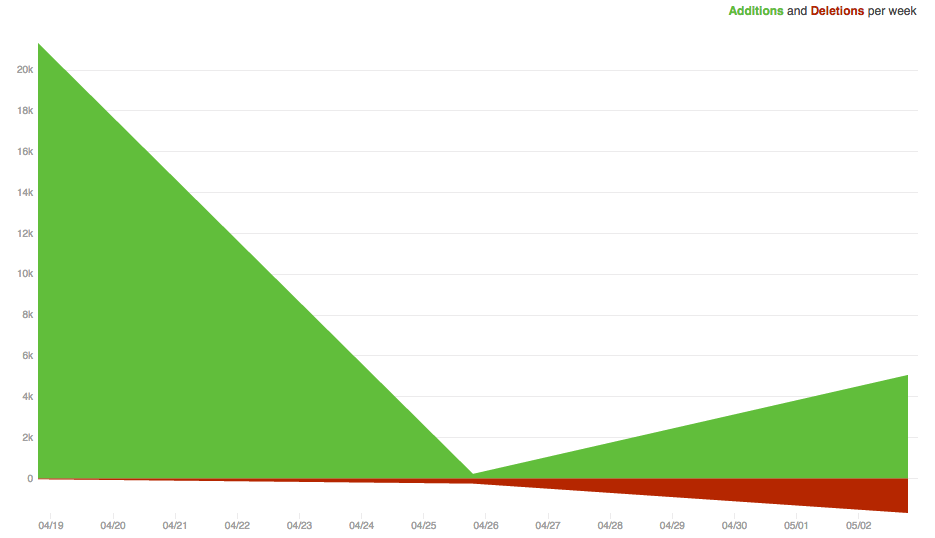
Gráfica de las contribuciones del master branch



Commits Realizados



Frecuencia de Código



Reflexión sobre el compilador.

Victor Clemente

Se muestra claramente como son usados todos los conocimientos necesarios para ser un profesional de tecnologías, mas que desde el echo de uso, de su comprensión y su aplicación. El desarrollo de un lenguaje nuevo de programación permite ver desde un punto de vista mas cercano, el funcionamiento abstracto de los ordenadores, asimismo abre fronteras hacia otros horizontes de programación, porque una vez que se sabe como son las cosas, se conoce como cambiarlas.

Jorge Curiel

Decidimos tomar este proyecto como una oportunidad de aprendizaje, no solo por los conocimientos adquiridos durante el semestre, si no también para manejar un nuevo lenguaje de programación. En este caso Python. En lo personal, no había utilizado este lenguaje anteriormente. Fue un poco confuso acostumbrarme a la sintaxis, pero siento que las reglas de indexación son un elemento muy útil para una mejor comprensión del código. En cuanto al desarrollo del compilador, Python nos proporcionó algunas facilidades que otros lenguajes no nos hubieran permitido, haciendo que la parte de programación en sí no fuera tan elaborada como las ideas abstractas. De las cosas que más se me complicaron fue el manejo de memoria y la administración de la tabla de procedimientos, al no haber utilizado nada tan complejo, pero siento que estos topes de cabeza me ayudaron a mejorar mis habilidades de una manera que no esperaba.

# Descripción del lenguaje

## Nombre de el Lenguaje

Noobscript – “Todos podemos Programar”

## Descripción genérica de características del lenguaje

Noobscript nos permite, por medio de un léxico sencillo, la capacidad de realizar operaciones básicas, esto incluye la capacidad de crear estatutos de asignación, condición, ciclo y escritura. Estos realizados con una aritmética de tipos fácil de entender por el lenguaje diseñado que permite realizar operaciones aritméticas sin que el usuario final se pierda.

De igual manera, el lenguaje permite manejo de funciones, las cuales manejan tipos de variables como globales, locales o temporales dependiendo ya de la necesidad de el programador, asimismo el manejo de datos estructurados.

## Posibles errores

Noobscript al ser un lenguaje de bajo nivel y con motivos de educación, genera errores básicos basados en los lenguajes de programación convencionales iterativos.

Los posibles errores que pueden ocurrir durante la compilación de noobscript yacen principalmente en la sintaxis del lenguaje, muestra los errores mala sintaxis para que el programador esté consciente de lo que sucede y le permita mejorar en el entendimiento de su error fácilmente.

Esto puede incluir errores comunes como incompatibilidad de tipos, asignaciones o asociaciones equivocadas respecto a tipos y errores de sintaxis y semántica de acuerdo a los asociado con el código.

# Descripción del compilador

## Hardware y Software

Se considera como plataforma de desarrollo el lenguaje de programación python 3.4.3 , ya que es un lenguaje fácil de aprender y de igual manera hace hincapié en una sintaxis que favorezca a un código legible. Este, al ser un lenguaje de alto nivel, es orientado a objetos, interpretado e interactivo y esto se considera esto como base y filosofía del lenguaje que se pretende crear. Este, será desarrollado en el sistema operativo Mac 10.10.3 (Yosemite) que es el más actual y nos permite mayor capacidad de procesamiento para evitar errores de cualquier tipo.

## Descripción del análisis léxico

Para el análisis del léxico se utilizó PLY, que es una implementación pura en Python de las populares herramientas de construcción para compiladores lex y YACC. Su principal objetivo es mantenerse lo más fiel posible a la manera tradicional en que trabaja lex/yacc. Esto incluye el manejo de parsing del tipo LALR(1) así como el proveer input de validación, reportes de errores y diagnósticos. Es por esto, que se decidió utilizar PLY, pues al ya estar familiarizados con lex y yacc, la implementación fue mucho más sencilla.

A continuación se muestra la lista del léxico permitido por Noobscript.

Dígitos

D = {1,2,3,4,5,6,7,8,9}

Abecedario

L = {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}

Operadores

O = { + , - , \* , / , < , >, =, == }

Delimitadores

De = { ( , ) , [ , ] , ; , : , : ,. , “}”, “{“ }

Lo anterior fue procesado por distintas expresiones regulares con formato en python para su uso en PLY.

Ejemplo:

Identificadores: r'[a-zA-Z\_][a-zA-Z\_0-9]\*'

Constantes Decimales: r'\d+\.\d+'

Cosstantes Enteras: r'\d+'

Delimitadores: r'\('

r'\)'

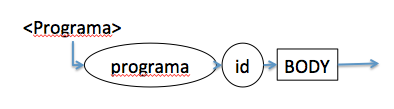
r'\: … y demás literales.'

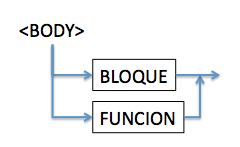
Palabras reservadas

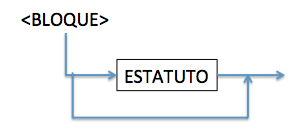
|  |  |
| --- | --- |
| **Palabra** | **Explicación - Uso** |
| Entero | Número Entero |
| Decimal | Número Decimal (float) |
| Frase | Cadena de caracteres |
| Imprime | Impresión en pantalla |
| Lee | Lectura de captura en pantalla |
| esVerdad | Valor verdadero (boolean) |
| Programa | Inicio del código de programación |
| Función | Inicio de una función |
| Mientras | Palabra para checar la expresión en do |
| Si | Inicio de una condición |

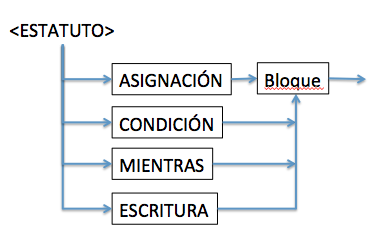
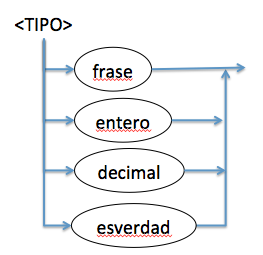
## Descripción del análisis de sintaxis

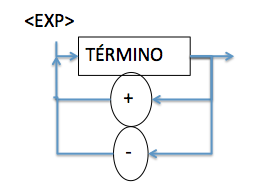
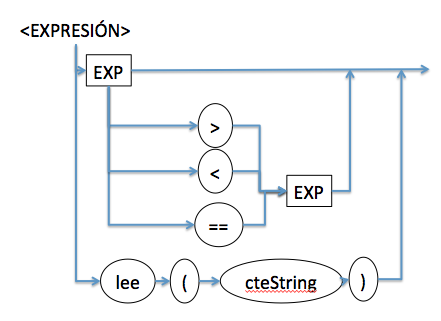
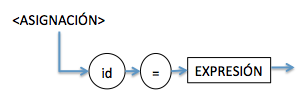
Para la sintaxis utilizamos los siguientes diagramas, los cuales convertimos en gramáticas formales bajo las convenciones requeridas por PLY (se muestran en el código anexo).

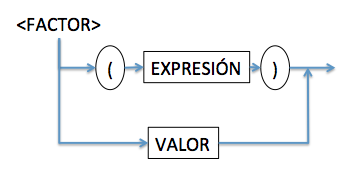
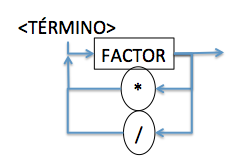


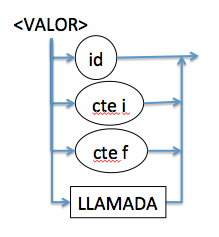


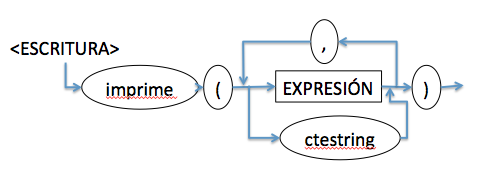
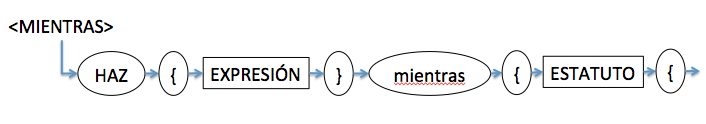
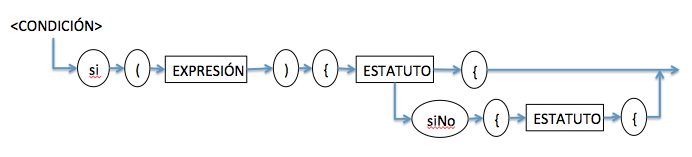
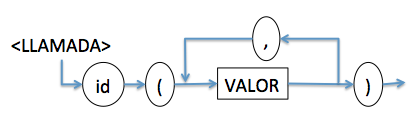












## Descripción del análisis semántico

Basamos las consideraciones semánticas en las operaciones básicas aritméticas restringiendo su compatibilidad respecto a los lenguajes más comunes para su aprendizaje.

Operadores

+ 1

- 2

\* 3

/ 4

< 5

> 6

== 7

= 8

( 9

) 10

GotoF 20

GotoV 21

Goto 22

ERA 30

Param 31

GOsub 32

Write 33

TEMPFRASE =6500

//constantes

CONSTENTERO = 7000

CONSTDECIMAL = 7500

CONSTesVerdad = 8000

CONSTFRASE = 8500

Tabla de consideraciones semánticas

entero + entero entero

decimal decimal

esVerdad E

frase E

entero - entero entero

decimal decimal

esVerdad E

frase E

entero \* entero entero

decimal decimal

esVerdad E

frase E

entero / entero decimal

decimal decimal

esVerdad E

frase E

entero = entero entero

decimal E

esVerdad E

frase E

entero == entero esVerdad

decimal esVerdad

esVerdad E

frase E

entero < entero esVerdad

> decimal esVerdad

esVerdad E

frase E

decimal + entero decimal

decimal decimal

esVerdad E

frase E

decimal - entero decimal

decimal decimal

esVerdad E

frase E

decimal \* entero decimal

decimal decimal

esVerdad E

frase E

decimal / entero decimal

decimal decimal

esVerdad E

frase E

decimal = entero decimal

decimal decimal

esVerdad E

frase E

decimal == entero esVerdad

decimal esVerdad

esVerdad E

frase E

decimal < entero esVerdad

> decimal esVerdad

esVerdad E

frase E

esVerdad + entero E

decimal E

esVerdad E

frase E

esVerdad - entero E

decimal E

esVerdad E

frase E

esVerdad \* entero E

decimal E

esVerdad E

frase E

esVerdad / entero E

decimal E

esVerdad E

frase E

esVerdad = entero E

decimal E

esVerdad esVerdad

frase E

esVerdad == entero E

decimal E

esVerdad esVerdad

frase E

esVerdad < entero E

> decimal E

esVerdad E

frase E

frase + entero E

decimal E

esVerdad E

frase E

frase - entero E

decimal E

esVerdad E

frase E

frase \* entero E

decimal E

esVerdad E

frase E

frase / entero E

decimal E

esVerdad E

frase E

frase = entero E

decimal E

esVerdad E

frase frase

frase == entero E

decimal E

esVerdad E

frase E

frase < entero E

> decimal E

== esVerdad E

frase E

Pseudo Código Cubo Semantico

char cubo[4][4][4 ]=

{

{

//Suma +

//entero + op2

//{entero,decimal,esVerdad,Frase}

{en, d, e, e},

//decimal + op2

{d, d, e, e},

//esVerdad + op2

{e, e, e, e},

//frase + op2

{e, e, e, e}

},

{

//Resta -

//entero - op2

{en, d, e, e},

//decimal - op2

{d, d, e, e},

//esVerdad - op2

{e, e, e, e},

//frase - op2

{e, e, e, e},

},

{

//Multiplicacion \*

//entero \* op2

{en, d, e, e},

//decimal \* op2

{d, d, e, e},

//esVerdad \* op2

{e, e, e, e},

//frase \* op2

{e, e, e, e},

},

{

//Division /

//entero / op2

{d, d, e, e},

//decimal / op2

{d, d, e, e},

//esVerdad / op2

{e, e, e, e},

//frase / op2

{e, e, e, e},

},

{

//Igualdad =

//int = op2

{en, en, e, e},

//decimal = op2

{d,d, e, e},

//esVerdad = op2

{e, e, es, e},

//frase = op2

{e, e, e, e}

},

{

//comparativo ==

//entero == op2

{es, es, e, e},

//decimal == op2

{es, es, e, e},

//esVerdad == op2

{e, e, es, e},

//frase == op2

{e, e, e, e}

},

{

//comparativos < >

//entero <> op2

{es, es, e, e},

//decimal <> op2

{es, es, e, e},

//esVerdad <> op2

{e, e, e, e},

//frase <> op2

{e, e, e, e}

},

};

# Descripción de la máquina virtual

## Equipo de computo

La máquina virtual fue desarrollada en el lenguaje Python de igual manera por su sencillez en administración de listas y lectura de archivos, lo cual permitió la fácil lectura de el archivo de cuádruplos .

## Descripción de el uso de memoria

//Globales

GLOBENTERO = 1000

GLOBDECIMAL =1500

GLOBesVerdad =2000

GLOBFRASE =2500

//Locales

LOCALENTERO =3000

LOCALDECIMAL =3500

LOCALesVerdad =4000

LOCALFRASE =4500

//Temporales

TEMPENTERO =5000

TEMPDECIMAL =5500

TEMPesVerdad =6000

# Listado con los principales módulos de el compilador

Maquina.py

Lee el archivo cuádruplos generado por el quad y realiza las operaciones de este código interpretado.

from local import \*

from temporal import \*

from constant import \*

from globales import \*

def checaScope(dire):

if dire >= 1000 and dire < 2500: #Direcciones de variables globales

return 1

elif dire >= 3000 and dire < 4500: #Direcciones de variables locales

return 2

elif dire >= 5000 and dire < 6500: #Direcciones de variables temporales

return 3

else:

return 4 #Regresa 4 si no es de ninguna de las otras memorias

#Metodo que determina el tipo de la variable que se usara. Si entra en alguno de los rangos especificados se regresa el tipo

#double o boolean (dependiendo del caso), sino se regresa el tipo int

def checaTipo(dire):

if dire == -1:

return -1 #//Si no hay una direccion en alguno de los elementos del cuadruplo

#rangos de variables float en las memorias

elif dire >= 1500 and dire < 2000 or dire >= 3500 and dire < 4000 or dire >= 5500 and dire < 6000 or dire >= 7500 and dire < 8000:

return 2

#Rangos de variables boolean en las memorias

elif dire >= 2000 and dire < 2500 or dire >= 4000 and dire < 4500 or dire >= 6000 and dire < 6500 or dire >= 8000 and dire < 8500:

return 3

#Rangos de variables string en las memorias

elif dire >= 2500 and dire < 3000 or dire >= 4500 and dire < 5000 or dire >= 6500 and dire < 7000 or dire >= 8500 and dire < 9000:

return 4

#si no pues es entero

else:

return 1

def obtenValorD(scope,dire, tipo):

if scope == 1:

return memglobal.getValD(dire, tipo)

elif scope == 2:

return memlocal.getValD(dire, tipo)

elif scope ==3:

return memtemp.getValD(dire, tipo)

else: #case 4:

return memconstant.getValD(dire, tipo)

def meteValorD(scope, dire, val, tipo):

if scope == 1:

memglobal.setValD(dire, val, tipo)

print (memglobal.getValD(dire, tipo))

elif scope == 2:

memlocal.setValD(dire, val, tipo)

elif scope == 3:

memtemp.setValD(dire, val, tipo)

else:

memconstant.setValD(dire, val, tipo)

def obtenValorS( scope, dire):

def uno(): # case 1:

return memglobal.getMemString(dire)

def dos(): #case 2:

return memlocal.getMemString(dire)

def tres(): # case 3:

return memtemp.getMemString(dire)

def cuatro(): #case 4 :

return memconstant.getMemString(dire)

operaciones = { 1: uno, 2: dos, 3: tres, 4: cuatro}

operaciones[scope]()

def meteValorS(scope, val, dire):

def uno():

memglobal.setMemString(dire, val)

def dos():

memlocal.setMemString(dire, val)

def tres():

memtemp.setMemString(dire, val)

def cuatro():

emconstant.setMemString(dire, val)

operaciones = { 1: uno, 2: dos, 3: tres, 4: cuatro}

operaciones[scope]()

def obtenValorB(scope, dire):

def uno():

return memglobal.getMemBool(dire)

def dos():

return memlocal.getMemBool(dire)

def tres():

return memtemp.getMemBool(dire)

operaciones = { 1: uno, 2: dos, 3: tres}

operaciones[scope]()

def meteValorB(scope, dire, val):

def uno() : #case 1:

memglobal.setMemBool(dire, val)

def dos():

memlocal.setMemBool(dire, val)

def tres():

memtemp.setMemBool(dire, val)

operaciones = { 1: uno, 2: dos, 3: tres}

operaciones[scope]()

#Inicializas Espacios en memoria con la clase correspondiente

memglobal = Global()

memlocal = Local()

memconstant = Constant()

memtemp = Temporal()

x\_table = []

cuad = [[]]

cont1 = 0

cont2 = 0

for eachLine in open('cuadruplos.txt', "r"):

cuad.append([int(k) for k in eachLine.split()])

cont2 = cont2 + 1

#El operador es el segundo elemento del arreglo obtenido

for e in range(len(cuad)-1):

cont1 = cont1 + 1

op = cuad[cont1][0]

scope = checaScope(cuad[cont1][1])

scope2 = checaScope(cuad[cont1][2])

scope3 = checaScope(cuad[cont1][3])

#Revisar el tipo de cada operando

tipo1 = checaTipo(cuad[cont1][1])

tipo2 = checaTipo(cuad[cont1][2])

tipo3 = checaTipo(cuad[cont1][3])

def suma(op): # case 1 +

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorD(scope3, cuad[cont1][3], (ope1 + ope2), tipo3)

def resta(op): # case 2: -

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorD(scope3, cuad[cont1][3], (ope1 - ope2), tipo3)

def multiplicacion(op): #case 3: \*

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorD(scope3, cuad[cont1][3], (ope1 \* ope2), tipo3)

def division(op): #case 4: /

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorD(scope3, cuad[cont1][3], (ope1 / ope2), tipo3)

def menor(op): #case 5: <

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorB(scope3, cuad[cont1][3], (ope1 < ope2))

def mayor(op): #case 6: >

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorB(scope3, cuad[cont1][3], (ope1 > ope2))

def comparacion(op): #case 7: ==

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

ope2 = obtenValorD(scope2, cuad[cont1][2], tipo2)

meteValorB(scope3, cuad[cont1][3], (ope1 == ope2))

def igualacion(op): #igucalacion case 8 =

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

def uno(tipo3):

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

meteValorD(scope3, cuad[cont1][3], ope1, tipo3) #cabiar ope 1

def dos(tipo3):

ope1 = obtenValorD(scope, cuad[cont1][1], tipo1)

meteValorD(scope3, cuad[cont1][3], ope1, tipo3)

def tres(tipo3):

ope1 = obtenValorB(scope, cuad[cont1][1], tipo1)

meteValorB(scope3, cuad[cont1][3], opeb)

def cuatro(tipo3):

opes = obtenValorS(scope, cuad[cont1][1], tipo1)

meteValorS(scope3, cuad[cont1][3], opes)

operaciones = { 1: uno, 2: dos, 3: tres, 4: cuatro}

operaciones[tipo3](tipo3)

def GOTOF(op): #case 20: //GOTOF

if obtenValorB(scope,cuad[cont1][1]) == True:

print ("siguele ")

else:

cont1 = cuad[cont1][3]

def GOTOV(op): #case 21: //GOTOV

if obtenValorB(scope,cuad[cont1][1]) == False:

print ("siguele ")

else:

cont1 = cuad[cont1][3]

def GOTO(op): #case 22: //GOTO

cont1 = cuad[cont1][3]

def ERA(op): #case 30: //ERA

print ("era")

def PARAM(op): #case 31: //PARAM

if tipo1 == 1 or tipo1 == 2:

val = obtenValorD(score, cuad[cont1][1], tipo1)

meteValorD(scope, cuad[cont1][3], val , tipo3)

elif tipo1== 3:

val = obtenValorB(score, cuad[cont1][1], tipo1)

meteValorB(scope, cuad[cont1][3], val , tipo3)

else:

val = obtenValorS(score, cuad[cont1][1], tipo1)

meteValorS(scope, cuad[cont1][3], val , tipo3)

def GOsub(op): #case 32: //Gosub

print("")

def write(op): #case 33: //Write

global scope

global scope2

global scope3

global tipo1

global tipo2

global tipo3

global cuad

def uno(tipo1): #case 1:

ope1 = memtemp.getValD(cuad[cont1][1])

print(ope1)

def dos(tipo1): #case 2:

ope1 = memtemp.getValD(cuad[cont1][1])

print(ope1)

def cuatro(tipo1):

ope1 = memtemp.getMemString(cuad[cont1][1])

print(ope1)

operaciones = { 1: uno, 2: dos, 4: cuatro}

operaciones[tipo1](tipo1)

operaciones = { 1: suma, 2: resta, 3: multiplicacion, 4: division, 5: menor, 6:mayor, 7:comparacion, 8:igualacion, 20:GOTOF, 21:GOTOV, 22:GOTO, 30:ERA, 31:PARAM, 32: GOsub, 33:write}

operaciones[op](op)

print("\n")

#--->cerrar archivo

#todo que devuelve el scope de un operando, recibe la direccion de la variables

#y determina la memoria a la que pertenece por medio de los rangos.

Cubo.py

Cubo semántico que realiza el chequeo de tipos para ver su si son compatibles y en caso de ser necesario regresa el tipo de la nueva variable temporal que será usada por quad.

#Cubo Semantico

tipo = -1

def suma(op, operando1 , operando2):

global tipo

print ("Estoy en el case suma Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 == 'es' or operando1=='f':

print ("no se puede")

return False

elif operando1 == 'e':

if operando2 == 'e':

print ("si se puede y es entero")

tipo = 5000

return True

elif operando2 == 'd':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es Decimal")

tipo = 5500

return True

elif operando2 == 'f':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

else:

return False

def resta(op, operando1 , operando2):

global tipo

print ("Estoy en el case resta Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 == 'f' or operando1 == 'es':

print ("Nel Carnal")

return False

elif operando1=='e':

if operando2 == 'e':

print ("si se puede y es Entero")

tipo = 5000

return True

elif operando2 =='d':

valido = True

tipo = 5500

print ("si se puede y es Decimal")

return valido

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es Decimal")

tipo = 5500

return True

elif operando2 == 'd':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

else:

print ("no se puede")

return False

def multiplicacion(op, operando1 , operando2):

global tipo

print ("Estoy en el case multiplicacion Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 == 'es' or operando1 == 'f':

print ("Nel Carnal")

return False

elif operando1 == 'e':

if operando2 =='e':

print ("si se puede y es Entero")

tipo = 5000

return True

elif operando2 == 'd':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'en':

print ("si se puede y es Decimal")

tipo = 5500

return True

elif operando2 == 'd':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

else:

print ("Nel vatirri")

return False

def division(op, operando1 , operando2):

global tipo

print ("Estoy en el case division! Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 =='es' or operando1 =='f':

print ("No se puede")

return False

elif operando1 =='e':

if operando2 == 'e':

print ("si se puede y es Decimal")

tipo = 5500

return True

elif operando2 == 'd':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es Decimal")

tipo = 5500

return True

elif operando2 =='d':

print ("si se puede y es Decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

else:

return False

def menorQue(op, operando1 , operando2):

print ("Estoy en el case en menorque <")

if operando1 =='es' or operando1 =='f':

print ("No se puede")

return False

if operando1 =='e':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 == 'd':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 =='d':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

else:

return False

def mayorQue(op, operando1 , operando2):

print ("Estoy en mayor que")

if operando1 =='es' or operando1 =='f':

print ("Nel Carnal")

return False

if operando1 =='e':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 == 'd':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 =='d':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

else:

return False

def igualA(op, operando1 , operando2):

print ("Estoy en el case relacional == ! Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 =='f':

return False

elif operando1 =='e':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 == 'd':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 == 'es':

print ("nel carnal")

return False

else:

print ("no se puede nigga")

return False

elif operando1 == 'd':

if operando2 == 'e':

print ("si se puede y es esVerdad")

tipo = 6000

return True

elif operando2 =='d':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

elif operando1 == 'es':

if operando2 == 'es':

print ("si se puede y es esVerdad")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

else:

return False

def igualacion(op, operando1 , operando2):

print ("Estoy en el case de igualacion Operando1:%s Operando2:%s\n" % (operando1, operando2))

if operando1 =='e':

if operando2 == 'e':

print ("si se puede y es entero")

return True

elif operando2 == 'd':

print ("si se puede y es decimal")

return True

else:

print ("no se puede nigga")

return False

elif operando1 =='d':

if operando2 == 'e':

print ("si se puede y es decimal")

tipo = 5500

return True

elif operando2 == 'd':

print ("si se puede y es decimal")

tipo = 5500

return True

else:

print ("no se puede nigga")

return False

elif operando1 =='es':

if operando2 == 'es':

print ("booleaaaaaan")

tipo = 6000

return True

else:

print ("no se puede nigga")

return False

elif operando1 =='f':

if operando2 == 'f':

print ("si se puede y es esFrase")

tipo = 6500

return True

else:

print ("no se puede nigga")

return False

else:

return False

def check(op, oper1 , oper2):

#entero

if oper1 >= 1000 and oper1 < 1500 or oper1 >= 3000 and oper1 < 3500 or oper1 >= 5000 and oper1 < 5500 or oper1 >= 7000 and oper1 < 7500:

operando1='e'

#decimal

elif oper1 >= 1500 and oper1 < 2000 or oper1 >= 3500 and oper1 < 4000 or oper1 >= 5500 and oper1 < 6000 or oper1 >= 7500 and oper1 < 8000:

operando1='d'

#esVerdad

elif oper1 >= 2000 and oper1 < 2500 or oper1 >= 4000 and oper1 < 4500 or oper1 >= 6000 and oper1 < 6500 or oper1 >= 8000 and oper1 < 8500:

operando1='es'

#frase

elif oper1 >= 2500 and oper1 < 3000 or oper1 >= 4500 and oper1 < 5000 or oper1 >= 6500 and oper1 < 7000 or oper1 >= 8500 and oper1 < 9000:

operando1='f'

#entero

if oper2 >= 1000 and oper2 < 1500 or oper2 >= 3000 and oper2 < 3500 or oper2 >= 5000 and oper2 < 5500 or oper1 >= 7000 and oper1 < 7500:

operando2='e'

#decimal

elif oper2 >= 1500 and oper2 < 2000 or oper2 >= 3500 and oper2 < 4000 or oper2 >= 5500 and oper2 < 6000 or oper1 >= 7500 and oper1 < 8000:

operando2='d'

#esVerdad

elif oper2 >= 2000 and oper2 < 2500 or oper2 >= 4000 and oper2 < 4500 or oper2 >= 6000 and oper2 < 6500 or oper1 >= 8000 and oper1 < 8500:

operando2='es'

#frase

elif oper2 >= 2500 and oper2 < 3000 or oper2 >= 4500 and oper2 < 5000 or oper2 >= 6500 and oper2 < 7000 or oper1 >= 8500 and oper1 < 9000:

operando2='f'

# Como si fuera switch, pero con arrays associativos

operaciones = { 1: suma, 2: resta, 3: multiplicacion, 4: division, 5: menorQue, 6: mayorQue, 7: igualA, 8:igualacion }

valido = operaciones[op](op, operando1, operando2)

return valido

#1000 entero

#1500 decimal

#2000 boolean

#2500 frase string

def tipotemp(): #regresa el tipo nuevo de la variable temporal que sera creada

global tipo

return tipo

quad.py

archivo de cuádruplos que crea el código interpretado para que sea ejecutado por la maquina virtual.

from cubo import check, tipotemp

cont=1

contemp=0

cuadruplos = [[0 for x in range(5)] for x in range(300)]

#Crear las pilas e inicializarlas

#pila de operadores

PilaO = [1001,1002] #pila de operandos

PSaltos = [] #pila de saltos

POper = [2] #pila de operadores

cuadf = None

def init():

# PilaO=createStack(100); #pila de operandos

# PSaltos=createStack(100); #pila de saltos

# POper=createStack(100); #pila de operadores

global cuadf

cuadf = open("cuadruplos.txt", 'w')

if cuadf == None:

print( "Archivo cuadruplos.txt no se ha podido abrir");

exit(1);

def printcuadruplos():

for i in range(300):

for j in range(5):

if cuadruplos[i][j]== 0:

exit(1)

global cuadf

cuadf.write(repr(cuadruplos[i][j]) + " ")

cuadf.write("\n") ;

#impresion de de direccion en memoria y de valor asignado de la variable dependiendo de su tipo

def escribe\_ctei(dire, val):

cuadf.write("%d %d %d\n",s,dire,val)

def escribe\_ctef(dire, val):

cuadf.write("%d %d %d\n",s,dire,val)

def escribe\_ctes(dire, val):

cuadf.write("%d %d %d\n",s,dire,val)

def escribe\_cuad(cuadro, ope, oper1, oper2, res):

print("Imprimiendo:", repr(cuadro),repr(ope), repr(oper1), repr(oper2), repr(res))

cuadruplos[cuadro-1][0] = cuadro

cuadruplos[cuadro-1][1] = ope

cuadruplos[cuadro-1][2] = oper1

cuadruplos[cuadro-1][3] = oper2

cuadruplos[cuadro-1][4] = res

global cont

cont = cont + 1

#1

def pila\_id(oper):

PilaO.append(oper)

print( "Ya meti oper \n", repr(oper))

#2

def pila\_op(op):

POper.append(op)

print( "ya meti op \n", repr(op))

#3

def parentesisPush():

POper.append(9) #genera pared falsa en pila

def parentesisPop():

POper.pop #saca de la pila

#5 ARREGLAR

def termino():

global contemp

print( "Estoy en cuadruplo termino\n")

op = POper[-1]

POper.pop()

if op == 3 or op == 4: #operadores \* o /

oper2 = PilaO[-1]

PilaO.pop()

oper1 = PilaO[-1]

PilaO.pop()

if (check(op, oper1, oper2)): #checa si es valido

escribe\_cuad(cont, op, oper1, oper2, contemp+tipotemp()) #genera el cuadruplo

pila\_id(contemp+tipotemp()) #mete el resultado a la pila de operadores

contemp = contemp+1

else: #marca error si no es compatible

print( "\*\*\*ERROR DE TIPOS\*\*\*\* termino\n")

exit(1)

else:

fprintf( "No es termino pasar al siguiente \n")

pila\_op(op)

#6

def expresion():

global contemp

print( "Estoy en cuadruplo expresion osea suma y resta")

op = POper[-1]

POper.pop()

if op == 1 or op == 2: #operadores + o -

oper2 = PilaO[-1]

PilaO.pop()

oper1 = PilaO[-1]

PilaO.pop()

if check(op, oper1, oper2): #checa el tipo

escribe\_cuad(cont, op, oper1, oper2, contemp+tipotemp()) #genera el cuadruplo

pila\_id(contemp+tipotemp()) #mete el resultado a la pila de operadores

contemp = contemp+1

else: #marca error si no es compatible

print( "\*\*\*ERROR DE TIPOS\*\*\*\* expresion\n")

exit(1)

else:

print( "No es expresion pasar al siguiente \n")

pila\_op(op)

#7

def relacional():

print( "Estoy en cuadruplo relacional\n ");

global contemp

op = POper[-1]

POper.pop()

if op == 5 or op == 6 or op == 7 : # op == < || op == > || op = ==

oper2 = PilaO[-1]

PilaO.pop()

oper1 = PilaO[-1]

PilaO.pop()

if check(op, oper1, oper2): #checa el tipo

escribe\_cuad(cont, op, oper1, oper2, contemp+tipotemp())

pila\_id(contemp+tipotemp()); #mete el resultado a la pila de operadores

contemp = contemp + 1

else: #marca error si no es compatible

print( "\*\*\*ERROR DE TIPOS\*\*\*\* relacional\n");

exit(1)

else:

print( "No es relacional pasar al siguiente \n");

pila\_op(op)

#8

def assign():

global contemp

print("Estoy en quad assign\n")

op = POper[-1]

POper.pop()

if op == 8: #si el operador es =

oper2 = PilaO[-1]

PilaO.pop()

oper1 = PilaO[-1]

PilaO.pop()

if check(op, oper1, oper2): #checa el tipo

escribe\_cuad(cont, op, oper2, -1, oper1) #genera el cuadruplo

contemp = contemp + 1

else: #marca error si no es compatible

print( "Error de tipos en assign\n")

exit(1)

else:

print( "No es asignacion pasar al siguiente \n")

pila\_op(op)

#15

def logico():

global contemp

print( "Estoy en cuad logico\n");

op = POper[-1]

POper.pop()

if op == 8 or op == 9: # op == & || op == |

oper2 = PilaO[-1]

PilaO.pop()

oper1 = PilaO[-1]

PilaO.pop()

if check(op, oper1, oper2): #checa el tipo

escribe\_cuad(cont, op, oper1, oper2, (contemp+tipotemp()))

pila\_id(contemp+tipotemp()) # mete el resultado a la pila de operadores

contemp = contemp + 1

else: #marca error si no es compatible

print( "\*\*\*ERROR DE TIPOS\*\*\*\* logico\n")

exit(1)

else:

print( "No es logico, pasar al siguiente \n")

pila\_op(op)

#10

def if1():

global cont

print( "Estoy en cuadruplo if\n")

#if

#1.- genera gotoF y mete cont-1 a la pila de saltos

op = 20

oper1 = PilaO[-1]

if per1 < 10000 or oper1 >= 11000:

print ("\*\*\*ERROR DE TIPOS\*\*\*\* IF\n") #STANDAR ERROR DE PYTHON CHECAR

else:

POper.pop()

escribe\_cuad(cont, op, oper1, -1, -1)

PSaltos.append(cont-1)

#11

def else1():

global cont

#2genera goto (else), y saca falso de la pila y rellena el primer salto

op = 22 #goto

escribe\_cuad(cont, op, -1, -1, -1)

falso = PSaltos[-1]

cuadruplos[falso-1][4]=cont

PSaltos.append(cont-1)

#12

def if2():

fin = PSaltos[-1]

PSaltos.pop()

cuadruplos[fin-1][4]=cont

#13

def do1():

#do-while

#1.- mete cont a pila de saltos

PSaltos.append(cont)

#14

def do2():

#2.- genra gotoV e incrementa el cont

op = 21 #gotov

oper1 = Pila0[-1]

oper2 = -1

escribe\_cuad(cont, op, oper1, -1, contemp)

def print1():

op=33

POper.append(op)

def print2():

oper1=top(PilaO)

escribe\_cuad(cont,top(POper),oper1,-1,-1)

def cuadproc():

return cont

init()

expresion()

PilaO.append(1)

POper.append(5001)

expresion()

printcuadruplos()

Constant.py, temporal.py, globales.py, local.py

Clases que sirven para obtener o meter valores en las direcciones de memoria respecto a su tipo, scope y lo que ordene la maquina virtual. (se anexa solo global.py por motivos de muestra)

class Global:

#enteros = 1000

#flotantes = 1500

#esverdad = 2000

#frases = 2500

memInt = [None]\*500

memFloat = [None]\*500

memBool = [None]\*500

memString = [None]\*500

def setValD(self, dire, val, tipo):

if tipo == 1:

dirReal = dire - 1000

if dirReal < 500:

self.memInt[dirReal] = val

elif tipo == 2:

dirReal = dire - 1500

if dirReal < 500:

memFloat[dirReal] = val

def getValD (self, dire, tipo):

if tipo == 1:

dirReal = dire - 1000

if dirReal < 500:

return self.memInt[dirReal]

elif tipo == 2:

dirReal = dire - 1500

if dirReal < 500:

return self.memFloat[dirReal]

def setMemBool(self, dire, val):

global esVerdad

dirReal = dire - 2000

if dirReal < 500:

memBool[dirReal] = self.val

def getMemBool(self, dire):

global esVerdad

dirReal = dire - 2000

if dirReal < 500:

return self.memBool[dirReal]

def setMemString(self, dire, val):

global frases

dirReal = dire - 25000

if dirReal < 500:

memString[dirReal] = self.val

def getMemString(self ,dire):

global frases

dirReal = dire - 2500

if dirReal < 500:

return self.memString[dirReal]

# Bibliografía

Welcome to Python.org. (n.d.). Retrieved March 4, 2015, from https://www.python.org/

Tutorials Point Simply Easy Learning. (n.d.). Retrieved March 2, 2015, from <http://www.tutorialspoint.com/python/index.htm>

Viswanathan, B. (2014, January 18). Quora. Retrieved March 1, 2015, from <http://www.quora.com/Why-is-programming-and-coding-so-hard>

The way of the program. (n.d.). Retrieved March 2, 2015, from <http://www.openbookproject.net/thinkcs/python/english2e/ch01.html>