**maquinaVirtual.py**

from acoustaticAlter3 import constantes

from cuadruplo import \*

from acoustaticAlter3 import CuadruploPila

import sys

from memoryAcoustatic import \*

import copy

sys.setrecursionlimit(4190)

#declaracion de memorias

memoriaGlobal = memoryAcoustatic()

memoriaLocal = memoryAcoustatic()

memoriaConstante = []

memoriaTemporal = []

#declaracion de memoria preparada para meterse a la pila (llamada de era)

memoriaHolding = memoryAcoustatic()

#declaracion de espacio de memoria que sostiene los params

params = {}

#declaracion de pila de cuadruplo siguiente pentiente

nextQuad = []

#declaracion de pila de memorias dormidas

sleepingMemories = []

#pila donde se guarda el offset de la variable global que corresponde al metodo que fue llamado. De esta manera, al momento del return, es posible hacer referencia a esta

sleepingOffsets = []

sleepingValues = []

#declaracion del lugar que contiene a los cuadruplos

cuadruploArray = {}

cuadruploActual = 0

#cantidad de temporales

tempQuantity = 0

#metodo en el cual me encuentro

actualContext = 13000

#Datos de Metodos - nombre de metodo | dir

pfile=open("vpm.obj","r")

data = pfile.read()

pfile.close()

lista = data.split("\n")

listaspliteada = []

for item in lista:

listaspliteada.append(item.split("|"))

#pila de memorias locales

pilaDeEjecucion = []

def quadsAllocation():

global tempQuantity

#print("########################################################")

#abrir el archivo

pfile=open("acoustatic.obj","r")

data = pfile.read()

pfile.close()

lista = data.split('\n')

#recorrer cada linea

for item in lista:

if(item != ""):

if(item[0] == 't'):

tempQuantity = int(item[1:])

else:

#splitear la linea

lineaActual = item.split('|')

#primer elemento es el id

numeroCuadruplo = int(lineaActual[0])

#guardar lo demas en un cuadruplo

cuadruploActual = cuadruplo(lineaActual[1],lineaActual[2],lineaActual[3],lineaActual[4])

#print(numeroCuadruplo)

cuadruploArray[numeroCuadruplo] = cuadruploActual

#print("########################################################")

def verifyCurrentQuad():

if(cuadruploActual >= len(cuadruploArray)):

# print("Hemos Terminado")

# print("memoriaConstante: ", memoriaConstante)

# print("memoriaTemporal: ", memoriaTemporal)

# print("memoriaLocalInt ", memoriaLocal.memoriaInt, "\n memoriaLocalFloat ", memoriaLocal.memoriaFloat, "\n memoriaLocalString ", memoriaLocal.memoriaString)

sys.exit()

def alocateGlobalVariables():

#averiguar cuantos espacios vamos a ocupar

#leer e interpretar la primera linea del archivo vpm.obj

punteros = listaspliteada[0]

metodo = int(listaspliteada[0][0])

cantInt =int(listaspliteada[0][1])

cantFloat =int(listaspliteada[0][2])

cantBool =int(listaspliteada[0][3])

cantString =int(listaspliteada[0][4])

cantSonido =int(listaspliteada[0][5])

cantInstrumento =int(listaspliteada[0][6])

cantPista =int(listaspliteada[0][7])

memoriaGlobal.initialize(cantInt, cantFloat, cantBool, cantString, cantSonido, cantInstrumento, cantPista)

#inicializar en cero los espacios correspondientes

def startLocalVariables(p):

#tengo que recorrer cada uno de los renglones de listaParametros

#inicializar memoriaLocal del renglon en el cual me encuentro

#averiguar cuantos espacios vamos a ocupar

#leer e interpretar la primera linea del archivo vpm.obj

p = int(p - 13000)

punteros = listaspliteada[p]

metodo = int(listaspliteada[p][0])

cantInt =int(listaspliteada[p][1])

cantFloat =int(listaspliteada[p][2])

cantBool =int(listaspliteada[p][3])

cantString =int(listaspliteada[p][4])

cantSonido =int(listaspliteada[p][5])

cantInstrumento =int(listaspliteada[p][6])

cantPista =int(listaspliteada[p][7])

memoriaLocal.initialize(cantInt, cantFloat, cantBool, cantString, cantSonido, cantInstrumento, cantPista)

def initializeConstants():

global memoriaConstante

pfile=open("constantes.obj","r")

data = pfile.read()

pfile.close()

lista = data.split("\n")

listaspliteada = []

constantKeys = []

#constant memory allocation

for i in range (0, len(lista)-1):

memoriaConstante.append('')

for item in lista:

listaspliteada.append(item.split("|"))

for i in range (0, len(listaspliteada)-1):

pointerConstants = int(listaspliteada[i][2])-14500

if(listaspliteada[i][0] == '0'):

memoriaConstante[pointerConstants] = (0, int(listaspliteada[i][1]))

elif(listaspliteada[i][0] == '1'):

memoriaConstante[pointerConstants] = (1, float(listaspliteada[i][1]))

elif(listaspliteada[i][0] == '2'):

if(listaspliteada[i][1] == 'on' or listaspliteada[i][1] == 'true'):

listaspliteada[i][1] = True

else:

listaspliteada[i][1]= False

memoriaConstante[pointerConstants] =(2, listaspliteada[i][1])

elif(listaspliteada[i][0] == '3'):

memoriaConstante[pointerConstants] =(3, listaspliteada[i][1][1:-1])

else:

print("ERROR DE TIPO")

print(listaspliteada[i])

sys.exit()

def alocateTemporal():

for i in range(tempQuantity):

memoriaTemporal.append("")

def saveTemporal(offset, value):

if(offset > 17500):

offset = offset - 17500

memoriaTemporal[offset] = value

def getType(direccion):

if(direccion >= 1000 and direccion < 2500):

return 0

elif(direccion >= 2500 and direccion < 4000):

return 1

elif(direccion >= 4000 and direccion < 5500):

return 2

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

return 3

elif(direccion >= 7000 and direccion < 8500):

return 4

elif(direccion >= 8500 and direccion < 10000):

return 5

elif(direccion >= 10000 and direccion < 11500):

return 6

elif(direccion >= 11500 and direccion < 13000):

return 7

elif(direccion >= 14500 and direccion < 17500):

offset = direccion - 14500

return memoriaConstante[offset][0]

else:

print("No es una direccion de un tipo de variable")

return -1

def setContent(direccion, valor):

global memoriaLocal

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

memoriaLocal.saveInt(offset, int(valor))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

memoriaLocal.saveFloat(offset, float(valor))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

if(valor == 'on' or valor == 'true' or valor==True):

memoriaLocal.saveBoolean(offset, True)

else:

memoriaLocal.saveBoolean(offset, False)

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = int(direccion - 5500)

memoriaLocal.saveString(offset,str(valor))

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("graba Sonido local")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("graba Instrumento")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("graba Pista")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("graba Pieza")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("graba Procedimiento local")

elif(direccion >= 14500 and direccion < 17500):

#es una constante

offset = direccion - 14500

#memoriaConstante[offset][1] = valor

memoriaConstante[offset] = valor

elif(direccion >= 17500 and direccion < 18000):

#es una temporal

offset = direccion - 17500

#print("este es el valor", offset, valor)

memoriaTemporal[offset] = valor

#print(memoriaTemporal)

elif(direccion >= 18000):

#es una variable global

direccion = direccion - 18000

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

memoriaGlobal.saveInt(offset, int(valor))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

memoriaGlobal.saveFloat(offset, float(valor))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

if(valor == 'on' or valor == 'true' or valor==True):

memoriaGlobal.saveBoolean(offset, True)

else:

memoriaGlobal.saveBoolean(offset, False)

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = direccion - 5500

memoriaGlobal.saveString(offset,str(valor))

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("graba Sonido global")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("graba Instrumento global")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("graba Pista global")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("graba Pieza global")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("graba Procedimiento local")

else:

print(direccion)

print("You are just amazing.....")

else:

print("Variable en lugar no permitido")

def getContent(direccion):

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

return int(memoriaLocal.getInt(offset))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

return float(memoriaLocal.getFloat(offset))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

respuesta = memoriaLocal.getBoolean(offset)

if(respuesta == 'on' or respuesta == 'true' or respuesta == True):

return True

else:

return False

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = direccion - 5500

return str(memoriaLocal.getString(offset))

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("regresa Sonido")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("regresa Instrumento")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("regresa Pista")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("regresa Pieza")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("regresa Procedimiento")

elif(direccion >= 14500 and direccion < 17500):

#es una constante

offset = direccion - 14500

return memoriaConstante[offset][1]

elif(direccion >= 17500 and direccion < 18000):

#es una temporal

offset = direccion - 17500

return memoriaTemporal[offset]

elif(direccion >= 18000):

#es una variable global

direccion = direccion - 18000

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

return int(memoriaGlobal.getInt(offset))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

return float(memoriaGlobal.getFloat(offset))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

respuesta = memoriaGlobal.getBoolean(offset)

if(respuesta == 'on' or respuesta == 'true' or respuesta == True):

return True

else:

return False

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = direccion - 5500

return str(memoriaGlobal.getString(offset))[1:-1]

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("regresa Sonido global")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("regresa Instrumento global")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("regresa Pista global")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("regresa Pieza global")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("regresa Procedimiento global somehow")

else:

print("ENSERIO, COMO LLEGASTE AQUI?!")

print(direccion)

else:

print("Variable en lugar no permitido")

def getContentParam(direccion):

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

return int(memoriaHolding.getInt(offset))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

return float(memoriaHolding.getFloat(offset))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

respuesta = memoriaHolding.getBoolean(offset)

if(respuesta == 'on' or respuesta == 'true'):

return True

else:

return False

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = direccion - 5500

return str(memoriaHolding.getString(offset))[1:-1]

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("regresa Sonido")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("regresa Instrumento")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("regresa Pista")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("regresa Pieza")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("regresa Procedimiento")

elif(direccion >= 14500 and direccion < 17500):

#es una constante

offset = direccion - 14500

return memoriaConstante[offset][1]

elif(direccion >= 17500 and direccion < 19000):

#es una temporal

offset = direccion - 17500

return memoriaTemporal[offset]

elif(direccion >= 19000):

#es una variable global

direccion - direccion - 19000

if(direccion >= 1000 and direccion < 2500):

#es un entero local

offset = direccion - 1000

return int(memoriaGlobal.getInt(offset))

elif(direccion >= 2500 and direccion < 4000):

#es un flotante local

offset = direccion - 2500

return float(memoriaGlobal.getFloat(offset))

elif(direccion >= 4000 and direccion < 5500):

#es un Booleano local

offset = direccion - 4000

respuesta = memoriaGlobal.getBoolean(offset)

if(respuesta == 'on' or respuesta == 'true'):

return True

else:

return False

elif(direccion >= 5500 and direccion < 7000):

#es un String logal

offset = direccion - 5500

return str(memoriaGlobal.getString(offset))[1:-1]

elif(direccion >= 7000 and direccion < 8500):

#es un Sonido local

print("regresa Sonido global")

elif(direccion >= 8500 and direccion < 10000):

#es un Instrumento local

print("regresa Instrumento global")

elif(direccion >= 10000 and direccion < 11500):

#es una pista local

print("regresa Pista global")

elif(direccion >= 11500 and direccion < 13000):

#es una pieza local

print("regresa Pieza global")

elif(direccion >= 13000 and direccion < 14500):

#es un procedimiento

print("regresa Procedimiento global somehow")

else:

print("ENSERIO, COMO LLEGASTE AQUI?!")

else:

print("Variable en lugar no permitido")

def operation(x):

global cuadruploActual, CuadruploPila, cuadruploArray

# print("CUADRUPLO"

# print("MEMORIA LOCAL")

# print("MEMORIA GLOBAL")

# memoriaGlobal.printMem()

# print("CONSTANTES")

# print(constantes)

# print("######################")

if(x == 0):

printer("NADA")

elif(x == 1):

printer("SUMA")

cuadruploSUMA()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 2):

printer("RESTA")

cuadruploRESTA()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 3):

printer("DIVISION")

cuadruploDIVISION()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 4):

printer("MULTIPLICACION")

cuadruploMULTIPLICACION()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 5):

printer("MENORQUE")

cuadruploMENORQUE()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 6):

printer("MENOROIGUAL")

cuadruploMENOROIGUAL()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 7):

printer("MAYORQUE")

cuadruploMAYORQUE()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 8):

printer("MAYOROIGUAL")

cuadruploMAYOROIGUAL()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 9):

printer("DIFERENTE")

cuadruploDIFERENTE()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 10):

printer("IGUALIGUAL")

cuadruploIGUALIGUAL()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 11):

printer("AND")

cuadruploAND()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 12):

printer("OR")

cuadruploOR()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 13):

printer("NOT")

cuadruploNOT()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 14):

printer("IGUALACION")

cuadruploIGUALACION()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 15):

printer("RESIDUO")

cuadruploRESIDUO()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 20):

printer("GOTO")

cuadruploGOTO()

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 21):

printer("GOTOV")

cuadruploGOTOV()

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 22):

printer("GOTOF")

cuadruploGOTOF()

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 23):

printer("GOSUB")

cuadruploGOSUB()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 24):

printer("ERA")

cuadruploERA()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 25):

printer("RETURN")

#significa que regresa algo

cuadruploRETURN()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 26):

printer("RET")

#solamente hace un salto

cuadruploRET()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 27):

printer("PARAM")

cuadruploPARAM()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

elif(x == 28):

printer("VER")

cuadruploVER()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

elif(x == 29):

printer("PRINT")

cuadruploPRINT()

cuadruploActual += 1

verifyCurrentQuad()

operation(int(cuadruploArray[cuadruploActual].op))

#

else:

printer("NOT SUCH METHOD")

cuadruploActual +=1

def parentesisCheck(p):

cuadruploATratar = p

#(isinstance(x, basestring)

if(not isinstance(cuadruploATratar.opd1, int) ):

if('(' in cuadruploATratar.opd1):

cuadruploATratar.opd1 = getContent(int(cuadruploATratar.opd1[1:-1]))

if(not isinstance(cuadruploATratar.opd2, int) ):

if('(' in cuadruploATratar.opd2):

cuadruploATratar.opd2 = getContent(int(cuadruploATratar.opd2[1:-1]))

if(not isinstance(cuadruploATratar.res, int) ):

if('(' in cuadruploATratar.res):

cuadruploATratar.res = getContent(int(cuadruploATratar.res[1:-1]))

return cuadruploATratar

def parentesisCheckParam(p):

cuadruploATratar = p

if(not isinstance(cuadruploATratar.opd1, int) ):

if('(' in cuadruploATratar.opd1):

cuadruploATratar.opd1 = getContentParam(int(cuadruploATratar.opd1[1:-1]))

if(not isinstance(cuadruploATratar.opd2, int) ):

if('(' in cuadruploATratar.opd2):

cuadruploATratar.opd2 = getContentParam(int(cuadruploATratar.opd2[1:-1]))

if(not isinstance(cuadruploATratar.res, int) ):

if('(' in cuadruploATratar.res):

cuadruploATratar.res = getContentParam(int(cuadruploATratar.res[1:-1]))

return cuadruploATratar

def cuadruploSUMA():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#parentesis significa tomar el valor de la temporal como direccion

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno + operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploRESTA():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno - operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploDIVISION():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno / operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploMULTIPLICACION():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno \* operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploMENORQUE():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno < operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploMENOROIGUAL():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno <= operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploMAYORQUE():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno > operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploMAYOROIGUAL():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno >= operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploDIFERENTE():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno != operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploIGUALIGUAL():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno == operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploAND():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno and operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploOR():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = operandoUno or operandoDos

setContent(direccionAGuardar, resultado)

def cuadruploNOT():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = getContent(int(cuadruploATratar.opd1))

#operandoDos = getContent(int(cuadruploATratar.opd2))

direccionAGuardar = int(cuadruploATratar.res)

resultado = not operandoUno

setContent(direccionAGuardar, resultado)

def cuadruploIGUALACION():

global cuadruploArray, cuadruploActual, memoriaLocal

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = int(cuadruploATratar.opd1)

#operandoDos = cuadruploATratar.opd2

direccionAGuardar = int(cuadruploATratar.res)

resultado = getContent(operandoUno)

setContent(direccionAGuardar, resultado)

def cuadruploRESIDUO():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

cuadruploATratar = parentesisCheck(cuadruploATratar)

operandoUno = int(cuadruploATratar.opd1)

operandoDos = cuadruploATratar.opd2

direccionAGuardar = cuadruploATratar.res

resultado = operandoUno % operandoDos

setContent(int(direccionAGuardar), resultado)

def cuadruploGOTO():

global cuadruploArray, cuadruploActual, memoriaLocal

if(int(cuadruploArray[cuadruploActual].opd1) != -1):

startLocalVariables(int(cuadruploArray[cuadruploActual].opd1))

cuadruploActual = int(cuadruploArray[cuadruploActual].res)

def cuadruploGOTOV():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

if(getContent(int(cuadruploATratar.opd1))):

cuadruploActual = int(cuadruploArray[cuadruploActual].res)

else:

cuadruploActual += 1

def cuadruploGOTOF():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

if(not getContent(int(cuadruploATratar.opd1))):

cuadruploActual = int(cuadruploArray[cuadruploActual].res)

else:

cuadruploActual += 1

def cuadruploGOSUB():

global cuadruploArray, cuadruploActual, sleepingMemories, memoriaHolding, nextQuad, params, memoriaLocal, sleepingValues

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#guardar en una pila el Siguiente cuadruplo

nextQuad.append(cuadruploActual + 1)

#guardar en la pila de variables pendientes la var que corresponde a este metdodo

sleepingOffsets.append(int(cuadruploATratar.opd2))

#guardar el sobre la que guardaremos todo lo que

#obtener cuadruplo objetivo 23 -1 variableGlobalQueRepresentaAlMetodo cuadruploAIr

pointers = [0,0,0,0,0,0,0,0]

# print(params)

# print(len(params))

#verificar si existe una variable global sobre la cual asignar un resultado

i = 1

while(i <= len(params)):

value = params[i]

valueType = value[1]

content = value[0]

varType = int(value[2])

direccionFinal = 1000 + (1500 \* varType + pointers[varType])

setContent(direccionFinal, content)

pointers[varType] += 1

i+=1

params = {}

pointers = [0,0,0,0,0,0,0,0]

cuadruploActual = int(cuadruploATratar.res)

# operation(20)

#operation(20)

#

def cuadruploERA():

global cuadruploArray, cuadruploActual, memoriaHolding, memoriaLocal

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

memoriaHolding = copy.deepcopy(memoriaLocal)

#guardar la memoria local en la pila de memorias

sleepingMemories.append(memoriaHolding)

memoriaLocal = copy.deepcopy(memoryAcoustatic())

startLocalVariables(int(cuadruploATratar.res))

def cuadruploRETURN():

global cuadruploArray, cuadruploActual, sleepingOffsets, sleepingMemories, sleepingValues

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#25 -1 -1 raviableARegresar

#sacar valor de variable a regresar

valor = getContent(int(cuadruploATratar.res))

#asignar este valor guardado a la variable global que corresponde al metodo

variableGlobal = int(sleepingOffsets.pop())

#print("VARGLOBL: ", variableGlobal)

# if(len(sleepingValues) > 0):

# holder = sleepingValues.pop()

# sleepingValues.append(valor)

# sleepingValues.append(holder)

# else:

# sleepingValues.append((variableGlobal,valor))

setContent(variableGlobal, valor)

sleepingOffsets.append(variableGlobal)

def cuadruploRET():

global cuadruploArray, cuadruploActual, nextQuad, sleepingMemories, sleepingOffsets, memoriaLocal

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#mata memorias y se las trae

# print("##")

# memoriaLocal.printMem()

memoryHolder = sleepingMemories.pop()

#mata memorias y se las trae

memoriaLocal = copy.deepcopy(memoryHolder)

# memoriaLocal.printMem()

# print("###")

memoryHolder = copy.deepcopy(memoryAcoustatic())

sleepingOffsets.pop()

cuadruploActual = int(nextQuad.pop())

def cuadruploPARAM():

global cuadruploArray, cuadruploActual

#numero 27 direccionDeVariableEnviando -1 numeroDeParametro

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#sacar el contenido de la direccion, tiene que ser de la memoria en holding

cuadruploATratar = parentesisCheckParam(cuadruploATratar)

value = getContentParam(int(cuadruploATratar.opd1))

type = getType(int(cuadruploATratar.opd1))

#guardarlo en el espacio necesario en mi arreglo de parametros

params[int(cuadruploATratar.res)] = (value, int(cuadruploATratar.opd1), type)

#el gosub es el que se encarga de agarrar los valores de ese arreglo y asignarlos a la nueva memoria

def cuadruploVER():

global cuadruploArray, cuadruploActual

#28 direccionDeNumeroAComparar limiteInferior limiteSuperior

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

numero = getContent(int(cuadruploATratar.opd1))

lInferior = int(cuadruploATratar.opd2)

lSuperior = int(cuadruploATratar.res)

if(not ((numero >= lInferior) and (numero <= lSuperior))):

print("Error de Fuera de Limite: ", numero)

sys.exit()

def cuadruploPRINT():

global cuadruploArray, cuadruploActual

cuadruploATratar = copy.deepcopy(cuadruploArray[cuadruploActual])

#AQUI ANTES TENIA PARENTESISCHECK PARAM

cuadruploATratar = parentesisCheck(cuadruploATratar)

if(int(cuadruploATratar.res) == -1):

print("")

else:

print(getContent(int(cuadruploATratar.res)))

#print(getContent(int(cuadruploATratar.res)), cuadruploATratar.res)

def printer(p):

i = 0

i += 1

def testQuads():

global cuadruploArray

quadAct = 0

while(quadAct < len(cuadruploArray)):

print(cuadruploArray[quadAct].op, '~', cuadruploArray[quadAct].opd1, '~', cuadruploArray[quadAct].opd2, '~', cuadruploArray[quadAct].res)

quadAct += 1

#inicia la ejecucion e inicializacion

initializeConstants()

alocateGlobalVariables()

#Cuadruplos

quadsAllocation()

alocateTemporal()

cuadruploActual = 0

operation(int(cuadruploArray[cuadruploActual].op))