**AcoustaticY.py**

import ply.yacc as yacc

import acoustaticGamma

import pprint

import cuboSemantico

import sys

from cuadruplo import cuadruplo

pp = pprint.PrettyPrinter(indent=4)

tokens = acoustaticGamma.tokens

#PILAS

#Pila de Operandos

PilaO = []

#Pila de Operadores

POper = []

#Pila de Saltos

PSaltos = []

#Pila Dimensiones

PilaDimensionadas = []

#Ultima variable invocada

lastvar = ""

#Dice si la ultima var fue constante

lastvarcte = 0

#Dice ultimo offset de cte

lastoffcte = 0

#Ultimo numero

lastnumber = 0

#Ultimo sonido declarado

lastSound = ""

#Ultimo instrumento declarado

lastInstr = ""

#dice el tipo de la ultima constante

casterType = 0

#Dice ultimo metodo llamado

lastMet = ""

#Contador de Parametros en metodo

contParam = 0

#Espacio de Registros Temporales Debe de ser contador?

Avail = 0

#Pila de Cuadruplos

CuadruploPila = []

#lista que contiene el orden de parametros cuando se declara un metodo

listaParametros = []

#Returns

PilaRet = []

#declaracion de offset

offsetEntero = 1000

offsetFlotante = 2500

offsetBooleano = 4000

offsetString = 5500

offsetSonido = 7000

offsetInstrumento = 8500

offsetPista = 10000

offsetPieza = 11500

offsetProcedimiento = 13000

offsetConstantes = 14500

offsetTemporales = 17500

#declaracion de punteros

#0 = entero, 1=flotante, 2=booleano, 3=string, 4=sonido, 5=instrumento, 6=pista, 7=pieza

punteros=[0,0,0,0,0,0,0,0]

#estructura de tupla

#(ID, tipo, numeroIDentificador, procedimientoAlQuePertenece)

#tabla en la cual se guardan todas las variables, en realidad es un diccionario. El primer nivel dicta a que procedimiento pertece.

#el segundo nivel es para la variable en si

variables = {}

#creamos la nueva dimension de la nueva direccion

variables[ offsetProcedimiento ] = {}

#tabla donde se guardan las constantes. Su composicion es diferente ya que no tienen ID

#(tipo, valor,numeroIDentificador)

constantes = {}

#tabla donde se guardan las constantes

#(tipo, valor, numeroIDentificador)

temporales = {}

#Lista que lleva el conteo de cuantas variables se declararon

numeroVariablesPorProcedimiento={}

#Lista de parametros de un procedimietno

parametrosProcedimiento = {}

parametrosProcedimiento[offsetProcedimiento] = {}

#lista de metodo con su offsetCorrespondiente

listaMetodos = {}

#Arreglo donde se guardan

test = [0,0,0,0]

start = 'piece'

def p\_piece(p):

'''piece : cuadruploPlay ACOUSTATIC PIECE ID OPENCORCH content CLOSECORCH'''

f = open('acoustatic.obj','w')

line = 't' + str(offsetTemporales - 17500)+ "\n"

f.write(line)

for i in range (0, len(CuadruploPila)):

#line = str(i) + ":" + str(CuadruploPila[i].op) + " " + str(CuadruploPila[i].opd1) + " " + str(CuadruploPila[i].opd2) + " " + str(CuadruploPila[i].res) + "\n"

line = str(i) +'|' + str(CuadruploPila[i].op) + '|' + str(CuadruploPila[i].opd1) + '|' + str(CuadruploPila[i].opd2) + '|' + str(CuadruploPila[i].res) + "\n"

f.write(line)

#saca variables

#saca constantes

f = open('constantes.obj','w')

listaKeys = constantes.keys()

for item in listaKeys:

linea = str(constantes[item][0])+"|"+str(constantes[item][1])+"|"+str(constantes[item][2])+"\n"

f.write(linea)

#saca variables Por procedimiento/metodo

f = open('vpm.obj','w')

listaKeys = numeroVariablesPorProcedimiento.keys()

for item in listaKeys:

if(len(numeroVariablesPorProcedimiento[item]) != 0):

linea = str(item)+'|'+str(numeroVariablesPorProcedimiento[item][0])+'|'+str(numeroVariablesPorProcedimiento[item][1])+'|'+str(numeroVariablesPorProcedimiento[item][2])+'|'+str(numeroVariablesPorProcedimiento[item][3])+'|'+str(numeroVariablesPorProcedimiento[item][4])+'|'+str(numeroVariablesPorProcedimiento[item][5])+'|'+str(numeroVariablesPorProcedimiento[item][6])+'|'+str(numeroVariablesPorProcedimiento[item][7])+"\n"

f.write(linea)

#saca la lista de metdodo/offset

f = open('metodoOffset.obj','w')

listaKeys = listaMetodos.keys()

for item in listaKeys:

linea = str(item) + '|' + str(listaMetodos[item][0])+ '|' + str(listaMetodos[item][1])+"\n"

f.write(linea)

def p\_cuadruploPlay(p):

'''cuadruploPlay : '''

CuadruploPila.append(cuadruplo(20, -1, -1 , ''))

def p\_content(p):

'''content : declaration content

| declaration'''

def p\_declaration(p):

'''declaration : NEW soundDeclaration

| NEW instrumentDeclaration

| NEW trackDeclaration

| assignment

| print'''

def p\_print(p):

'''print : PRINT OPENPAREN CLOSEPAREN SEMICOLON creaPrint1

| PRINT OPENPAREN printContent CLOSEPAREN SEMICOLON creaPrint2'''

def p\_creaPrint1(p):

'''creaPrint1 : '''

CuadruploPila.append(cuadruplo(29, -1, -1, -1))

def p\_creaPrint2(p):

'''creaPrint2 : '''

pr = PilaO.pop()

CuadruploPila.append(cuadruplo(29, -1, -1, pr))

def p\_printContent(p):

'''printContent : casterString STRINGLINE meteCte meteID quitaCte COMMA printContent

| casterString STRINGLINE meteCte meteID quitaCte

| idornum COMMA printContent

| idornum

| ID meteArr OPENBRACK fondoFalso exp2 creaVer CLOSEBRACK creaCuadsArr'''

def p\_soundAttribute(p) :

'''soundAttribute : intensityAttribute SEMICOLON soundAttribute

| durationAttribute SEMICOLON soundAttribute

| toneAttribute SEMICOLON soundAttribute

| pitchAttribute SEMICOLON'''

def p\_soundDeclaration(p):

'''soundDeclaration : SOUND ID insertSonido meteSonido OPENCORCH soundAttribute CLOSECORCH

| SOUND ID insertSonido SEMICOLON'''

def p\_insertSonido(p):

'''insertSonido : '''

global lastSound

insertVariableEsp((p[-1], 4, offsetSonido+punteros[4], 4))

lastSound = p[-1]

def p\_meteSonido(p):

'''meteSonido : '''

global PilaDimensionadas, lastSound

id = lastSound

PilaDimensionadas.append((id, getID(id)[4], getID(id)[2]))

def p\_intensityAttribute(p):

'''intensityAttribute : INTENSITY COLON exp2 '''

global PilaO, PilaDimensionadas

op1 = PilaO.pop()

dir = PilaDimensionadas[len(PilaDimensionadas)-1][2]

CuadruploPila.append(cuadruplo(14, op1, -1 , dir))

def p\_pitchAttribute(p):

'''pitchAttribute : PITCH COLON exp2'''

global PilaO, PilaDimensionadas

op1 = PilaO.pop()

dir = int(PilaDimensionadas.pop()[2]) + 3

CuadruploPila.append(cuadruplo(14, op1, -1 , dir))

def p\_durationAttribute(p):

'''durationAttribute : DURATION COLON exp2'''

global PilaO, PilaDimensionadas

op1 = PilaO.pop()

dir = int(PilaDimensionadas[len(PilaDimensionadas)-1][2]) + 1

CuadruploPila.append(cuadruplo(14, op1, -1 , dir))

def p\_toneAttribute(p):

'''toneAttribute : TONE COLON exp2'''

global PilaO, PilaDimensionadas

op1 = PilaO.pop()

dir = int(PilaDimensionadas[len(PilaDimensionadas)-1][2]) + 2

CuadruploPila.append(cuadruplo(14, op1, -1 , dir))

def p\_instrumentDeclaration(p):

'''instrumentDeclaration : INSTRUMENT ID insertInstrument OPENCORCH soundsPart CLOSECORCH

| INSTRUMENT ID insertInstrument'''

def p\_insertInstrument(p):

'''insertInstrument : '''

global lastInstr

#El tamano es -1 porque no se sabe cuantos atributos tendra

insertVariableEsp((p[-1], 5, offsetInstrumento+punteros[5], -1))

lastInstr = p[-1]

def p\_soundsPart(p):

'''soundsPart : SOUNDS OPENCORCH soundInnerPart CLOSECORCH'''

def p\_soundInnerPart(p):

'''soundInnerPart : ID meteInstr1 COMMA soundInnerPart

| NEW soundDeclaration meteInstr2 COMMA soundInnerPart

| dynamicSoundCreation COMMA soundInnerPart

| ID meteInstr1

| NEW soundDeclaration

| dynamicSoundCreation'''

def p\_meteInstr1(p):

'''meteInstr1 : '''

global punteros

id = getID(p[-1])

if(id == 'NONE'):

print ("Sonido: ", p[-1], " no inicializado")

sys.exit()

elif(id[1] != 4):

print ("Variable: ", p[-1]," no es tipo Sonido")

else:

CuadruploPila.append(cuadruplo(14, id[2], -1, offsetInstrumento+punteros[5]))

punteros[5] += 1

def p\_meteInstr2(p):

'''meteInstr2 : '''

global PilaDimensionadas, lastSound

id = lastSound

tupla = getID(id)

CuadruploPila.append(cuadruplo(14, tupla[2], -1, offsetInstrumento+punteros[5]))

punteros[5] += 1

def p\_dynamicSoundCreation(p):

'''dynamicSoundCreation : CREATE ID '\_' 'X' OPENCORCH dynamicHelper1 CLOSECORCH COLON exp2 COLON exp2'''

##AUN TENGO DUDAS AQUI

def p\_dynamicHelper1(p):

'''dynamicHelper1 : NEW soundDeclaration

| ID'''

def p\_trackDeclaration(p):

'''trackDeclaration : INSTRUMENTS OPENCORCH trackDeclarationHelper CLOSECORCH playTrack mainMethod

| INSTRUMENTS OPENCORCH trackDeclarationHelper CLOSECORCH methodBlock playTrack mainMethod'''

def p\_mainMethod(p):

'''mainMethod : newMethodCreation FUNCTION VOID MAIN methodListAdd fillMain OPENPAREN methodDeclatationParameter CLOSEPAREN OPENCORCH method CLOSECORCH'''

newMethod(p[1])

def p\_fillMain(p):

'''fillMain : '''

CuadruploPila[0].res = len(CuadruploPila)

CuadruploPila[0].opd1 = listaMetodos["main"][0]

def p\_methodBlock(p):

'''methodBlock : methodDeclaration methodBlock

| methodDeclaration'''

def p\_returnStatement(p):

'''returnStatement : RETURN idornum creaReturn SEMICOLON

| RETURN methodCallingParam creaReturn SEMICOLON'''

def p\_creaReturn(p):

'''creaReturn : '''

global PilaO, CuadruploPila

ret = PilaO.pop()

CuadruploPila.append(cuadruplo(25, -1, -1, ret))

def p\_methodDeclaration(p):

'''methodDeclaration : newMethodCreation FUNCTION type ID methodListAdd OPENPAREN methodDeclatationParameter CLOSEPAREN addParams OPENCORCH method CLOSECORCH meterReturn'''

def p\_addParams(p):

'''addParams : '''

parametrosProcedimiento[offsetProcedimiento] = listaParametros

def p\_meterReturn(p):

'''meterReturn : '''

CuadruploPila.append(cuadruplo(26, -1, -1, -1))

def p\_methodListAdd(p):

'''methodListAdd : '''

listaMetodos[p[-1]] = (offsetProcedimiento, len(CuadruploPila))

puntero = numeroVariablesPorProcedimiento[13000][test[1]]

if(test[1] == 0):

dir = offsetEntero + puntero

elif(test[1] == 1):

dir = offsetFlotante + puntero

elif(test[1] == 2):

dir = offsetBooleano + puntero

elif(test[1] == 3):

dir = offsetString + puntero

dir = dir + 18000

numeroVariablesPorProcedimiento[13000][test[1]] += 1

variables[13000][p[-1]] = (p[-1],test[1],dir,13000,0)

def giveMeParameter(p):

#p es el id del procedimiento :)

try:

return parametrosProcedimiento[p]

except KeyError:

return "NONE"

def p\_methodDeclatationParameter(p):

'''methodDeclatationParameter : methodPair COMMA methodDeclatationParameter

| methodPair

|'''

def p\_methodPair(p):

'''methodPair : methodType ID meterVariable'''

def p\_methodType(p):

'''methodType : SOUND

| BOOLEAN

| INSTRUMENT

| TRACK

| PIECE

| INT

| FL

| STRING'''

global test

listaParametros.append(p[1])

if(p[1] == "boolean"):

test[1] = 2

test[2] = offsetBooleano+punteros[2]

elif(p[1] == "string"):

test[1] = 3

test[2] = offsetString+punteros[3]

elif(p[1] == 'int'):

test[1] = 0

test[2] = offsetEntero+punteros[0]

elif(p[1] == "fl"):

test[1] = 1

test[2] = offsetFlotante+punteros[1]

def p\_newMethodCreation(p):

'''newMethodCreation : '''

newMethod(p)

def p\_trackDeclarationHelper(p):

'''trackDeclarationHelper : ID COMMA trackDeclarationHelper

| NEW instrumentDeclaration COMMA trackDeclarationHelper

| ID

| NEW instrumentDeclaration'''

def p\_playTrack(p):

'''playTrack : PLAY OPENCORCH playTrackHelper CLOSECORCH'''

newMethod(p[1])

listaMetodos["play"] = (offsetProcedimiento, len(CuadruploPila))

def p\_playTrackHelper(p):

'''playTrackHelper : timeSector playTrackHelper

| timeSector

| methodCalling playTrackHelper

| methodCalling'''

def p\_timeSector(p):

'''timeSector : exp2 COLON exp2 OPENCORCH timeSectorHelper CLOSECORCH'''

def p\_timeSectorHelper(p):

'''timeSectorHelper : ID SEMICOLON timeSectorHelper

| ID SEMICOLON

| ID COLON LOOPS SEMICOLON timeSectorHelper

| ID COLON LOOPS SEMICOLON

| ID COLON LOOPS OPENCORCH timeSectorHelper2 CLOSECORCH SEMICOLON timeSectorHelper

| ID COLON LOOPS OPENCORCH timeSectorHelper2 CLOSECORCH SEMICOLON'''

# | ID COLON LOOPS OPENCORCH timeSectorHelper2 CLOSECORCH localConditional SEMICOLON timeSectorHelper

# | ID COLON LOOPS OPENCORCH timeSectorHelper2 CLOSECORCH localConditional SEMICOLON

# | ID COLON LOOPS localConditional SEMICOLON timeSectorHelper

# | ID COLON LOOPS localConditional SEMICOLON'''

def p\_timeSectorHelper2(p):

'''timeSectorHelper2 : ID OFF COMMA timeSectorHelper2

| ID OFF'''

insertConstant((2, false))

def p\_methodCalling(p):

'''methodCalling : ID verificaMetodo OPENPAREN creaEra methodParameter CLOSEPAREN SEMICOLON creaSub

| ID verificaMetodo OPENPAREN verificaContParam CLOSEPAREN SEMICOLON creaSub'''

def p\_verificaMetodo(p):

'''verificaMetodo : '''

global lastMet

if(getMethodID(p[-1]) == "NONE"):

print ("Metodo: ", p[-1], " no inicializado")

sys.exit()

else:

lastMet = p[-1]

def p\_creaEra(p):

'''creaEra : '''

global lastMet, contParam

CuadruploPila.append(cuadruplo(24, -1, -1, listaMetodos[lastMet][0]))

contParam = 1

def p\_creaSub(p):

'''creaSub : '''

global lastMet

numCuadruplo = listaMetodos[lastMet][1]

CuadruploPila.append(cuadruplo(23, -1, variables[13000][lastMet][2], numCuadruplo))

def p\_methodCallingParam(p):

'''methodCallingParam : ID verificaMetodo OPENPAREN creaEra methodParameter CLOSEPAREN creaSub

| ID verificaMetodo OPENPAREN verificaContParam CLOSEPAREN creaSub'''

meterPilaO(p[1])

def p\_methodParameter(p):

'''methodParameter : depthHelperMethod COMMA sumaContParam methodParameter

| depthHelperMethod verificaContParam'''

def p\_sumaContParam(p):

'''sumaContParam : '''

global contParam

contParam += 1

def p\_verificaContParam(p):

'''verificaContParam : '''

global parametrosProcedimiento, contParam

#Trae el offset del Proc.

listaMet = getMethodID(lastMet)[0]

#Trae el numero de parametros

paramMet = len(parametrosProcedimiento[listaMet])

if(contParam != paramMet):

print("Numero de parametros de metodo ", lastMet, " no coincide con las declaradas")

sys.exit()

contParam = 0

def p\_depthHelper(p):

'''depthHelper : idornum

| idornum COLON depthHelper'''

def p\_depthHelperMethod(p):

'''depthHelperMethod : idornumParam meteParam

| idornumParam COLON depthHelperMethod'''

def p\_meteParam(p):

'''meteParam : '''

global PilaO

argumento = PilaO.pop()

#<--Verifica si tipo de argumento corresponde

if True:

CuadruploPila.append(cuadruplo(27, argumento, -1, contParam))

def p\_idornumParam(p):

'''idornumParam : ID meterVariable2 meteID

| casterInt INTEGER meteCte meteID quitaCte

| casterFloat FLOAT meteCte meteID quitaCte

| casterString STRINGLINE meteCte meteID quitaCte

| methodCallingParam'''

#aqui se necesita hacer la resolucion y agregacion de las constantes en acso de integer, float o string y la resolucion del ID

#aqui se tiene que hacer una llamada a un medoco como insertConstant, pero para method parameter

def p\_method(p):

'''method : sentence

| sentence method'''

def p\_sentence(p):

'''sentence : soundMod

| addSound

| toggleSound

| condition

| assignment

| arrayCreation

| cycle

| print

| methodCalling

| returnStatement'''

def p\_soundMod(p):

'''soundMod : depthHelper propertyAssignation SEMICOLON'''

def p\_propertyAssignation(p):

'''propertyAssignation : soundProperty COLON exp2'''

def p\_soundProperty(p):

'''soundProperty : INTENSITY

| PITCH

| TONE

| DURATION'''

def p\_addSound(p):

'''addSound : ADD ID COLON ID switch SEMICOLON

| ADD ID COLON NEW soundDeclaration SEMICOLON'''

#aun no se que hacer con switch

def p\_switch(p):

'''switch : ON

| OFF'''

global constantes, offsetConstantes

constantes[offsetConstantes] = (2, p[1], offsetConstantes)

offsetConstantes +=1

def p\_arrayCreation(p):

'''arrayCreation : type ID OPENBRACK num CLOSEBRACK SEMICOLON'''

global test, punteros, lastnumber

insertVariableArr((p[2], test[1], test[2], lastnumber))

def p\_type(p):

'''type : SOUND

| INSTRUMENT

| BOOLEAN

| TRACK

| PIECE

| INT

| FL

| VOID

| STRING'''

global test

if(p[1] == "boolean"):

test[1] = 2

test[2] = offsetBooleano+punteros[2]

elif(p[1] == "string"):

test[1] = 3

test[2] = offsetString+punteros[3]

elif(p[1] == 'int'):

test[1] = 0

test[2] = offsetEntero+punteros[0]

elif(p[1] == "fl"):

test[1] = 1

test[2] = offsetFlotante+punteros[1]

def p\_assignment(p):

'''assignment : simpleAssignment SEMICOLON

| ID meteArr OPENBRACK fondoFalso exp2 creaVer CLOSEBRACK creaCuadsArr EQUALS meteIg exp2 verificaIg SEMICOLON'''

def p\_meteArr(p):

'''meteArr : '''

global PilaDimensionadas

id = p[-1]

PilaDimensionadas.append((id, getID(id)[4], getID(id)[2]))

def p\_fondoFalso(p):

'''fondoFalso : '''

global POper

POper.append(p[-1])

def p\_creaVer(p):

'''creaVer : '''

global CuadruploPila, PilaO, PilaDimensionadas

Linf = 0

Lsup = int(PilaDimensionadas[len(PilaDimensionadas)-1][1]) - 1

tope = PilaO[len(PilaO)-1]

CuadruploPila.append(cuadruplo(28, tope, Linf , Lsup))

def p\_creaCuadsArr(p):

'''creaCuadsArr : '''

global CuadruploPila, PilaO, POper, offsetTemporales

aux = PilaO.pop()

T = offsetTemporales

dirBase = PilaDimensionadas.pop()[2]

insertConstant((0, dirBase))

base = getConstant(dirBase)

CuadruploPila.append(cuadruplo(1, aux, base , T))

PilaO.append("("+str(T)+")")

quitaFondoF = POper.pop()

offsetTemporales = offsetTemporales + 1

def p\_simpleAssignment(p):

'''simpleAssignment : type ID meterVariable meteID EQUALS meteIg typeAssigmentVerification

| ID meterVariable2 meteID EQUALS meteIg typeAssigmentVerification'''

def p\_typeAssigmentVerification(p):

'''typeAssigmentVerification : booleanExp verificaIg

| casterString STRINGLINE meteCte meteID quitaCte verificaIg

| exp2 verificaIg

| methodCallingParam verificaIg

| ID meteArr OPENBRACK fondoFalso exp2 creaVer CLOSEBRACK creaCuadsArr verificaIg'''

#tengo que usar test1 para el tipo

#insertVariable((p[2], test[1], test[2]))

def p\_quitaCte(p):

'''quitaCte : '''

global lastvarcte

lastvarcte = 0

def p\_meterVariable(p):

'''meterVariable : '''

global lastvar

insertVariable((p[-1], test[1], test[2]))

lastvar = p[-1]

def p\_exp2(p):

'''exp2 : t2 verificaSumRest

| t2 verificaSumRest PLUS metePlus exp2

| t2 verificaSumRest MINUS meteMinus exp2'''

#aqui va sienmpre a p1 por que siempre terminas aqui

def p\_t2(p):

'''t2 : idornum verificaMultDiv

| idornum verificaMultDiv MULT meteMult t2

| idornum verificaMultDiv DIV meteDiv t2

| idornum verificaMultDiv REM meteRem t2

| OPENPAREN exp2 CLOSEPAREN

| OPENPAREN exp2 CLOSEPAREN MULT meteMult t2

| OPENPAREN exp2 CLOSEPAREN DIV meteDiv t2

| OPENPAREN exp2 CLOSEPAREN REM meteRem t2'''

#print("t2");

def p\_num(p):

'''num : casterInt INTEGER meteCte meteID

| casterFloat FLOAT meteCte meteID'''

global lastvarcte, lastnumber

lastvarcte = 0

lastnumber = p[2]

def p\_casterInt(p):

'''casterInt : '''

global casterType

casterType = 0

def p\_casterString(p):

'''casterString : '''

global casterType

casterType = 3

def p\_casterFloat(p):

'''casterFloat : '''

global casterType

casterType = 1

def p\_meteCte(p):

'''meteCte : '''

global lastvarcte, lastoffcte, offsetConstantes, casterType

lastvarcte = 1

#print(casterType)

if(casterType == 0):

insertConstant((0, p[-1]))

elif (casterType == 1):

insertConstant((1, p[-1]))

else:

insertConstant((3, p[-1]))

lastoffcte = getConstant(p[-1])

def p\_cycle(p):

'''cycle : while

| doWhile

| for'''

def p\_while(p):

'''while : WHILE meterPSalto OPENPAREN booleanExp verificaOr CLOSEPAREN OPENCORCH verificaCiclo method CLOSECORCH sacaWhile'''

def p\_doWhile(p):

'''doWhile : DO meterPSalto OPENCORCH method CLOSECORCH WHILE OPENPAREN booleanExp CLOSEPAREN SEMICOLON verificaCiclo2'''

def p\_for(p):

'''for : FOR OPENPAREN simpleAssignment SEMICOLON booleanExp SEMICOLON simpleAssignment CLOSEPAREN OPENCORCH method CLOSECORCH'''

def p\_booleanExp(p):

'''booleanExp : TRUE meteCteBool meteID

| FALSE meteCteBool meteID

| exp2 GT meteGT exp2 verificaBool

| exp2 LT meteLT exp2 verificaBool

| exp2 GE meteGE exp2 verificaBool

| exp2 LE meteLE exp2 verificaBool

| exp2 EE meteEE exp2 verificaBool

| exp2 NE meteNE exp2 verificaBool

| exp2 GT meteGT exp2 verificaBool booleanComparison booleanExp verificaAnd

| exp2 LT meteLT exp2 verificaBool booleanComparison booleanExp verificaAnd

| exp2 GE meteGE exp2 verificaBool booleanComparison booleanExp verificaAnd

| exp2 LE meteLE exp2 verificaBool booleanComparison booleanExp verificaAnd

| exp2 EE meteEE exp2 verificaBool booleanComparison booleanExp verificaAnd

| exp2 NE meteNE exp2 verificaBool booleanComparison booleanExp verificaAnd'''

global lastvarcte

lastvarcte = 0

# try:

# print(p[2])

# except:

# insertConstant((2,p[1]))

def p\_meteCteBool(p):

'''meteCteBool : '''

global lastvarcte, lastoffcte, offsetConstantes

lastvarcte = 1

insertConstant((2,p[-1]))

lastoffcte = getConstant(p[-1])

def p\_booleanComparison(p):

'''booleanComparison : AND meteAnd

| OR meteOr

| NOT meteNE'''

def p\_condition(p):

'''condition : IF OPENPAREN booleanExp CLOSEPAREN OPENCORCH verificaCiclo method CLOSECORCH elseBlock'''

def p\_elseBlock(p):

'''elseBlock : ELSE sacaPSaltos2 OPENCORCH method CLOSECORCH sacaPSaltos

| sacaPSaltos'''

def p\_toggleSound(p):

'''toggleSound : depthHelper switch SEMICOLON'''

def p\_idornum(p):

'''idornum : ID lastVar meteID

| num'''

def p\_lastVar(p):

'''lastVar : '''

global lastvar

if(p[-1] != None):

lastvar = p[-1]

def getID(p):

try:

return variables[offsetProcedimiento][p]

except KeyError:

try:

return variables[13000][p]

except KeyError:

return "NONE"

def getMethodID(p):

try:

return listaMetodos[p]

except KeyError:

return "NONE"

#print("BARRERA")

#print(variables)

def p\_meteGT(p):

'''meteGT : '''

meterPOper(7)

def p\_meteLT(p):

'''meteLT : '''

meterPOper(5)

def p\_meteGE(p):

'''meteGE : '''

meterPOper(8)

def p\_meteLE(p):

'''meteLE : '''

meterPOper(6)

def p\_meteEE(p):

'''meteEE : '''

meterPOper(10)

def p\_meteNE(p):

'''meteNE : '''

meterPOper(9)

def p\_meteAnd(p):

'''meteAnd : '''

meterPOper(11)

def p\_meteOr(p):

'''meteOr : '''

meterPOper(12)

def p\_meterMult(p):

'''meteMult : '''

meterPOper(4)

def p\_meterDiv(p):

'''meteDiv : '''

meterPOper(3)

def p\_meterRem(p):

'''meteRem : '''

meterPOper(15)

def p\_metePlus(p):

'''metePlus : '''

meterPOper(1)

def p\_meteMinus(p):

'''meteMinus : '''

meterPOper(2)

def p\_meteIg(p):

'''meteIg : '''

meterPOper(14)

def p\_meteID(p):

'''meteID : '''

meterPilaO(lastvar)

def meterPilaO(opd):

#Verifica si ya tiene num asignado

if(lastvarcte == 1):

PilaO.append(lastoffcte)

#print (PilaO)

else:

meter = getID(opd)

if (meter == 'NONE'):

print ("Variable: ", opd, " no inicializada")

sys.exit()

else:

PilaO.append(meter[2])

def meterPOper(o):

global POper

POper.append(o)

def p\_verificaIg(p):

'''verificaIg : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if POper[len(POper)-1] == 14 :

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

#<--Verifica si el tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd2, -1, opd1))

def p\_verificaMultDiv(p):

'''verificaMultDiv : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if (POper[len(POper)-1] == 4) or (POper[len(POper)-1] == 3) or (POper[len(POper)-1] == 15):

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

res = offsetTemporales

#<--Verifica si el tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd1, opd2, res))

PilaO.append(res)

offsetTemporales = offsetTemporales + 1

def p\_verificaSumRest(p):

'''verificaSumRest : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if POper[len(POper)-1] == 1 or POper[len(POper)-1] == 2:

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

res = offsetTemporales

#<--Verifica si que tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd1, opd2, res))

PilaO.append(res)

offsetTemporales = offsetTemporales + 1

def p\_verificaBool(p):

'''verificaBool : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if POper[len(POper)-1] == 5 or POper[len(POper)-1] == 6 or POper[len(POper)-1] == 7 or POper[len(POper)-1] == 8 or POper[len(POper)-1] == 9 or POper[len(POper)-1] == 10 :

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

res = offsetTemporales

#<--Verifica si que tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd1, opd2, res))

PilaO.append(res)

offsetTemporales = offsetTemporales + 1

def p\_verificaAnd(p):

'''verificaAnd : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if POper[len(POper)-1] == 11 :

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

res = offsetTemporales

#<--Verifica si que tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd1, opd2, res))

PilaO.append(res)

offsetTemporales = offsetTemporales + 1

def p\_verificaOr(p):

'''verificaOr : '''

global POper, PilaO, offsetTemporales, CuadruploPila

if(len(POper)!=0):

if POper[len(POper)-1] == 12 :

op = POper.pop()

opd2 = PilaO.pop()

opd1 = PilaO.pop()

res = offsetTemporales

#<--Verifica si que tipo de var debe ser y si es correcto que se este haciendo la operacion

if True :

CuadruploPila.append(cuadruplo(op, opd1, opd2, res))

PilaO.append(res)

offsetTemporales = offsetTemporales + 1

def p\_verificaCiclo(p):

'''verificaCiclo : '''

global PSaltos, PilaO, CuadruploPila

#<--Verifica tipo de PilaO es boolean

if True :

resultado = PilaO.pop()

CuadruploPila.append(cuadruplo(22, resultado, -1, ""))

PSaltos.append(len(CuadruploPila) - 1)

def p\_verificaCiclo2(p):

'''verificaCiclo2 : '''

global PSaltos, PilaO, CuadruploPila

#<--Verifica tipo de PilaO es boolean

if True :

resultado = PilaO.pop()

fin = PSaltos.pop()

CuadruploPila.append(cuadruplo(21, resultado, fin, ""))

PSaltos.append(len(CuadruploPila) - 1)

def p\_sacaPSaltos(p):

'''sacaPSaltos : '''

global PSaltos, PilaO, offsetTemporales, CuadruploPila

fin = PSaltos.pop()

CuadruploPila[fin].res = len(CuadruploPila)

def p\_sacaPSaltos2(p):

'''sacaPSaltos2 : '''

global PSaltos, PilaO, offsetTemporales, CuadruploPila

CuadruploPila.append(cuadruplo(20, -1, -1, ""))

falso = PSaltos.pop()

CuadruploPila[falso].res = len(CuadruploPila)

PSaltos.append(len(CuadruploPila) - 1)

def p\_meterPSalto(p):

'''meterPSalto : '''

global PSaltos, PilaO, offsetTemporales, CuadruploPila

PSaltos.append(len(CuadruploPila))

def p\_sacaWhile(p):

'''sacaWhile : '''

global PSaltos, PilaO, offsetTemporales, CuadruploPila

falso = PSaltos.pop()

retorno = PSaltos.pop()

CuadruploPila.append(cuadruplo(20, -1, -1, retorno))

CuadruploPila[falso].res = len(CuadruploPila)

def p\_error(p):

print("Error de sintaxis en linea: ", p.lineno, " en: ", p.value)

# yacc.errok()

#0 = entero, 1=flotante, 2=booleano, 3=string,

#4=sonido, 5=instrumento, 6=pista, 7=pieza

def insertVariable(p):

if (getID(p[0]) == 'NONE'):

#recibe una tupla (parentesis) con todas las posiciones a meter en la tabla de variables

#(ID de variable p0

#tipo P1

#numeroIDentificador p2

global variables, punteros, offsetProcedimiento

#insertalo

#print("offset vale esto", offsetProcedimiento)

if(offsetProcedimiento == 13000):

dir = 18000 + p[2]

else:

dir = p[2]

variables[offsetProcedimiento][p[0]] = (p[0],p[1],dir,offsetProcedimiento,0)

#actualiza el puntero Correspondiente

punteros[p[1]] +=1

else:

print("Variable: ", p[0], "ya inicializada")

sys.exit()

def insertVariableArr(p):

if (getID(p[0]) == 'NONE'):

#recibe una tupla (parentesis) con todas las posiciones a meter en la tabla de variables

#(ID de variable p0

#tipo P1

#numeroIDentificador p2

global variables, punteros, offsetProcedimiento

#insertalo

#print("offset vale esto", offsetProcedimiento)

if(offsetProcedimiento == 13000):

dir = 18000 + p[2]

else:

dir = p[2]

variables[offsetProcedimiento][p[0]] = (p[0],p[1],dir,offsetProcedimiento, p[3])

#actualiza el puntero Correspondiente

punteros[p[1]] += int(p[3])

quitaPilaO = PilaO.pop()

else:

print("Variable: ", p[0], "ya inicializada")

sys.exit()

def insertVariableEsp(p):

if (getID(p[0]) == 'NONE'):

#recibe una tupla (parentesis) con todas las posiciones a meter en la tabla de variables

#(ID de variable p0

#tipo P1

#numeroIDentificador p2

global variables, punteros, offsetProcedimiento

dir = 18000 + p[2]

variables[13000][p[0]] = (p[0],p[1],dir,offsetProcedimiento, p[3])

#actualiza el puntero Correspondiente

punteros[p[1]] += int(p[3])

else:

print("Variable: ", p[0], "ya inicializada")

sys.exit()

def insertConstant(p):

#recibe una tupla (parentesis) con todas las posiciones a meter en la tabla de variables

#tipo P0

#valor p1

global constantes, offsetConstantes

if (getConstant(p[1]) == 'NONE'):

constantes[p[1]] = (p[0],p[1],offsetConstantes)

offsetConstantes += 1

def getConstant(p):

try:

return constantes[p][2]

except KeyError:

return "NONE"

def insertTemporal(p):

#recibe una tupla (parentesis) con todas las posiciones a meter en la tabla de variables

#tipo P0

#valor p1

#identificador) p2

global temporales, offsetTemporales

temporales[p[1]] = (p[0],p[1],offsetTemporales)

offsetTemporales += 1

def newMethod(p):

#guarda cantidad de variables declaradas

global listaParametros, variables, numeroVariablesPorProcedimiento, offsetProcedimiento, punteros, parametrosProcedimiento, listaMetodos

numeroVariablesPorProcedimiento[offsetProcedimiento] = punteros

#resetea punteros

punteros=[0,0,0,0,0,0,0,0]

#modifica offsetPRocedimiento

offsetProcedimiento += 1

#print("offsetTecnicamente aumento", offsetProcedimiento)

#guarda el nombre del nuevo metodo

#listaMetodos[p] = offsetProcedimiento

#agrega dimension en variables y en mumeroVariables

variables[offsetProcedimiento] = {}

numeroVariablesPorProcedimiento[offsetProcedimiento]= {}

parametrosProcedimiento[offsetProcedimiento] = {}

listaParametros = []

def dameIDVaraible(param):

variableAregresar = -1

#recibo el id en param

idATrabajar = param

#posteriormente necesito saber en que metodo me encuentro

metodoActual = offsetProcedimiento

#si no esta en el metodo actual, entonces necesito ir a las varaiblers globales

variableAregresar = [key for (key,value) in variables.items() if value[0] == idATrabajar]

#si esta, "regreso" en una variable el numero

def p\_meterVariable2(p):

'''meterVariable2 : '''

global lastvar

if (getID(p[-1]) == 'NONE'):

print ("Variable: ", opd, " no inicializada")

sys.exit()

else:

lastvar = p[-1]

pfile=open("manejoArreglo.txt","r")

data = pfile.read()

pfile.close()

yacc.yacc()

yacc.parse(data)