



Proyecto de Compiladores

**TUBY**

Ruben Alejandro Hernandez Gonzalez A01175209

Andres Marcelo Garza Cantu A00814236

23 de Noviembre de 2016

**Ing. Elda Qurioga**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Ruben Alejandro Andres Marcelo**

**Hernandez Gonzalez Garza Cantu**

[**a.1) Visión, Objetivos y Alcance del Proyecto**](#_lodlcwllwc29)[**4**](#_lodlcwllwc29)

[VISIÓN/PROPÓSITO DEL PROYECTO](#_rapar3xu1ef2) [4](#_rapar3xu1ef2)

[OBJETIVO DEL LENGUAJE CON ÁREA DE APLICACIÓN](#_hb29woho7d6u) [4](#_hb29woho7d6u)

[**a.2) Análisis de Requerimientos y Casos de Uso Generales**](#_5e3a6qlt73bh)[**4**](#_5e3a6qlt73bh)

[Requerimientos](#_5b8anw82rltv) [4](#_5b8anw82rltv)

[Casos de uso:](#_6nrm7dnc0z1x) [5](#_6nrm7dnc0z1x)

[**a.3) Descripción de los principales Test Cases**](#_1fk3dndaq8d6)[**5**](#_1fk3dndaq8d6)

[Test Cases](#_twek24o07d53) [5](#_twek24o07d53)

[**a.4) Descripción del proceso**](#_6xgf2memmgah)[**5**](#_6xgf2memmgah)

[Descripción del proceso general seguido para el desarrollo del proyecto](#_41jia5i2ger5) [5](#_41jia5i2ger5)

[Bitácora](#_ojwk4hefjqef) [5](#_ojwk4hefjqef)

[Reflexión Ruben Alejandro Hernandez Gonzalez](#_1xmldihzxjea) [7](#_1xmldihzxjea)

[Reflexión Andres Marcelo Garza Cantu](#_8bgl7kb2w21o) [7](#_8bgl7kb2w21o)

[**b.1) Nombre del Lenguaje**](#_y9dl1te1oryb)[**8**](#_y9dl1te1oryb)

[**b.2) Principales características del lenguaje**](#_mprk43qjzjof)[**8**](#_mprk43qjzjof)

[**b.3) Posibles errores de compilación/ejecución**](#_9rr2guel5qnl)[**8**](#_9rr2guel5qnl)

[Errores en compilacion](#_vngix1vlp4o2) [8](#_vngix1vlp4o2)

[Errores en ejecución](#_h4vwusd08zi6) [8](#_h4vwusd08zi6)

[**c.1) Equipo de computo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto**](#_nq9djzlkufic)[**8**](#_nq9djzlkufic)

[**c.2) Descripción de Análisis de Léxico.**](#_m8q33g6whaeo)[**8**](#_m8q33g6whaeo)

[Tokens asociados al lenguaje de programación](#_bcx75crk3dzi) [8](#_bcx75crk3dzi)

[**c.3) Descripción de Análisis de Sintaxis**](#_tf49vp42vtma)[**9**](#_tf49vp42vtma)

[**c.4) Descripción de Generación de Código Intermedio**](#_as5aoump88wf)[**22**](#_as5aoump88wf)

[Asignación](#_wy4h3jd3gxq1) [22](#_wy4h3jd3gxq1)

[Bloque](#_xsp978cwtilf) [23](#_xsp978cwtilf)

[Bloque-Clase](#_ce9s2a5bqnn6) [23](#_ce9s2a5bqnn6)

[Ciclo](#_57f2k1khfxsb) [24](#_57f2k1khfxsb)

[Clase](#_uaant23jnr93) [24](#_uaant23jnr93)

[Condición](#_bgvlb3v2w4kv) [24](#_bgvlb3v2w4kv)

[Declaracion](#_sfd3owygcgqs) [25](#_sfd3owygcgqs)

[Entrada](#_67jaa5ecazio) [25](#_67jaa5ecazio)

[Exp](#_9y4vqyth1e2) [26](#_9y4vqyth1e2)

[Expresa](#_797jq24mpst) [26](#_797jq24mpst)

[Expresión](#_l4trfjrm1eu) [26](#_l4trfjrm1eu)

[Factor](#_f626l79oj6wd) [27](#_f626l79oj6wd)

[Funcion](#_bzeatipwo123) [27](#_bzeatipwo123)

[Funcion-Principal](#_xudcsuoyajox) [27](#_xudcsuoyajox)

[Llamada-Función](#_mjjc7f5ywt6z) [28](#_mjjc7f5ywt6z)

[Llamada-IDs](#_v42hks2lil) [28](#_v42hks2lil)

[Programa](#_wkwv7zn0snp5) [29](#_wkwv7zn0snp5)

[Salida](#_2d84yix0vg54) [29](#_2d84yix0vg54)

[Terminal](#_70d8l5zgjd3n) [30](#_70d8l5zgjd3n)

[Termino](#_ymrp0emsbtn8) [30](#_ymrp0emsbtn8)

[Tipo](#_33j0g2m6c1cj) [31](#_33j0g2m6c1cj)

[Valor Salida](#_m8uruj54723q) [31](#_m8uruj54723q)

[**c.5) Descripción del proceso de Administración de memoria**](#_81o3e5lhi8kj)[**32**](#_81o3e5lhi8kj)

[**d.1) Equipo de computo, lenguaje y utilerías especiales utilizadas.**](#_suu81rs4s5q4)[**32**](#_suu81rs4s5q4)

[**d.2) Descripción detallada del proceso de Administración de Memoria en ejecución.**](#_a5dbekcf2ur8)[**32**](#_a5dbekcf2ur8)

[**e.1) Pruebas que comprueben el funcionamiento del lenguaje**](#_tz66xd4bxtey)[**33**](#_tz66xd4bxtey)

[Prueba con archivo propio hello.txt](#_yd40fdsdospv) [33](#_yd40fdsdospv)

[Prueba Fibonacci con valor n=10](#_lsfax5a5z8j3) [34](#_lsfax5a5z8j3)

[**Scanner.py**](#_wlvuidexw4bv)[**34**](#_wlvuidexw4bv)

[**maquina.py**](#_6d7ofqhrq9m4)[**56**](#_6d7ofqhrq9m4)

[**memoria.py**](#_v2p241wo59pb)[**65**](#_v2p241wo59pb)

[**tablas.py**](#_3ayxtcoe4cws)[**68**](#_3ayxtcoe4cws)

[Python](#_guhxb7p9g254) [73](#_guhxb7p9g254)

[PLY](#_pz9kqtg1cmxb) [73](#_pz9kqtg1cmxb)

[PyCharm](#_pec1b2wrl2zr) [73](#_pec1b2wrl2zr)

[**Identificadores**](#_y9qsvq2212eh)[**74**](#_y9qsvq2212eh)

[Variables numericas](#_rwol4xmop3pi) [74](#_rwol4xmop3pi)

[Variables de texto](#_xcg9oppgp886) [74](#_xcg9oppgp886)

[Variables Booleanas](#_dikc8kb4p041) [74](#_dikc8kb4p041)

[**Palabras reservadas**](#_v8xuzggfya9s)[**74**](#_v8xuzggfya9s)

[**Diseño de programas**](#_qixvy3jqi2mj)[**75**](#_qixvy3jqi2mj)

[**Expresiones**](#_5i82kx7buqlq)[**75**](#_5i82kx7buqlq)

[**Condiciones**](#_bdk0qymiylgf)[**75**](#_bdk0qymiylgf)

[**Ciclos**](#_7ammh2dbx2pw)[**75**](#_7ammh2dbx2pw)

[**Funciones**](#_fa0jt44nf9fc)[**76**](#_fa0jt44nf9fc)

[**Clases**](#_xba1dnmu6wjz)[**76**](#_xba1dnmu6wjz)

Documentación

a) Descripción del Proyecto

## a.1) Visión, Objetivos y Alcance del Proyecto

#### VISIÓN/PROPÓSITO DEL PROYECTO

Nuestra visión de este proyecto es el enseñarles a usuarios que hablan en español a familiarizarse con lógica computacional. Debido a que en la población mexicana, [[1]](#footnote-0) solo un 5%de la población entienden el lenguaje en inglés; esto se vuelve un impedimento al querer introducir estudiantes al área de la tecnología computacional.La barrera del lenguaje no deberia de existir dentro del area,computacional , con este proyecto buscamos cerrar esa brecha que no deberia existir. Sin embargo buscamos usar un lenguaje de alto nivel , orientado a objetos , debido a que si el usuario sabe manejar un lenguaje orientado a objetos , será capaz de adaptarse rápidamente a otros lenguajes

#### OBJETIVO DEL LENGUAJE CON ÁREA DE APLICACIÓN

Se desea implementar la lengua española a un lenguaje sencillo de programación. Con el propósito de poder facilitar el aprendizaje de la logica de programacion para los estudiantes que no cuenten con experiencia con el lenguaje inglés. Se busca que el lenguaje sea de alto nivel orientado a objetos,con el fin de que facilita la comprensión de programacion basica para alguien que no tenga experiencia con el área de ciencia de computación. El lenguaje devolverá una retroalimentación sencilla y fácil de comprender que facilite la codificación del programa. El lenguaje tendrá capacidaddes de manejo de clases , herencia , funciones especiales de clase , manejo de arreglo de una y dos dimensiones.

## a.2) Análisis de Requerimientos y Casos de Uso Generales

#### Requerimientos

* El lenguaje debe estar en español
* Debe ser orientado a objetos
* Ser un lenguaje de texto
* Debe correr en terminal

#### 

#### Casos de uso:

* Soporte de llamada de funciones.
* Soporte de herencia de clases
* Pueden llamarse funciones dentro de una clase.
* Soporte de operaciones aritméticas básicas
* Soporte de ciclos simples y condiciones

## a.3) Descripción de los principales Test Cases

#### Test Cases

* Factorial cíclico y recursivo
* Fibonacci ciclico y recursivo
* Find en un vector
* Sort en un vector
* Multiplicación de dos matrices
* Operaciones entre 2 funciones de clases

## a.4) Descripción del proceso

#### Descripción del proceso general seguido para el desarrollo del proyecto

El proyecto fue desarrollado utilizando python como el lenguaje de programación dedicado a implementar la lógica utilizada para desarrollar el programa. Dentro de Python utilizamos PLY que es una implementación de las herramientas para desarrollar compiladores Lex y yacc dentro de nuestro compilador. Y utilizamos Github para tener un control de versiones constante sobre los cambios realizados por cada integrante.

#### Bitácora

Septiembre/9/2016

Completar visión del proyecto, objetivo del lenguaje.

Septiembre/14/2016

Completar Léxico y Sintaxis del Lenguaje.

Septiembre/20/2016

Correcciones de sintaxis del Lenguaje

Septiembre/25/2016

Correccion de Tipo de lenguaje

Septiembre/29/2016

Se empezo a programar el lenguaje. Ultimas correcciones de sintaxis.

Octubre/1/2016

Se implemento Clase en la sintaxis del programa

Octubre/2/2016

Se implementaron Funciones, Ciclos, Declaracion de Variables, entre otras.

Octubre/6/2016

Se arreglaron errores de sintaxis, definicion de tokens

Octubre/11/2016

Se arreglo recursion infinita en Bloque y se agregaron tokens nuevos al programa.

Octubre/20/2016

Se empezo a verificar sintaxis de variables y se arreglaron errores de variables.

Octubre/27/2016

Se empezo a implementar las tablas de simbolos.

Noviembre/1/2016

Arreglos en la logica de las tablas de simbolos.

Noviembre/3/2016

Se arreglaron conflictos de sintaxis y se inicio el cubo semantico.

Noviembre/5/2016

Se realizo el diseño final del cubo semantico

Noviembre/7/2016

Se diseñaron las tablas para cuadruplos

Noviembre/8/2016

Se implementan los cambios a la tabla de cuadruplos, se empiezan a guardar valores.

Noviembre/9/2016

Se implementan verificacion de variables de clase y se implementan mas cuadruplos.

Noviembre/10/2016

Se implementan cuadruplos de Condicion y Mientras.

Noviembre/17/2016

Se implementaron temporales a salots de GoTo y se integraron llamadas de funciones al codigo.

Noviembre/18/2016

Se empieza a implementar la tabla de procedimientos, se guardan Goto de clases hacia la funcion principal y se insertan los RET de Funciones y clases. Se agregan las funciones de memoria.

Noviembre/19/2016

Se declara lógica de memoria y manejador de temporales. Se pueden meter funciones a la tabla de procedimientos, se pudieron realizar cambios a terminal. Ya se pudieron meter correctamente los ERA y Gosubs en los cuádruplos.

Noviembre/20/2016

Se implementó memoria en las las variables a guardar.

Noviembre/21/2016

Se empezó a implementar la máquina virtual y las direcciones de funciones y clases a en memoria.

Se empezó a escribir la documentación del compilador.

Noviembre/22/2016

Arreglos y matrices, direccionamientos en máquina virtual, se consideran constantes.

#### Reflexión Ruben Alejandro Hernandez Gonzalez

Esta fue una experiencia única, diría yo. Esta clase y proyecto fueron completamente diferente a lo que esperaba, fue pesado, pero cada vez que me equivocaba en algo, sentía que aprendía varias cosas mas. Este compilador me hace apreciar más lo que es el trabajo de diseñar un nuevo lenguaje de programación y del proceso que se tiene que tener al momento de considerar toda la logica detras de las funciones que quieres darle a tu lenguaje. El proyecto no se sintió bonito, pero se sentia bonito cada vez que logramos encontrar una solución a un problema que creíamos gigante anteriormente, asi que para mi fue una buena experiencia en cuestion de aprendizaje y satisfacción con uno mismo.

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

(Firma)

#### Reflexión Andres Marcelo Garza Cantu

Los conceptos de compilación de un lenguaje pueden variar bastante , manejo de memoria , asignación de matrices,, optimizacion de codigo y maquina virtual , son todos conceptos , que por el simple hecho de destacar en uno de estos ámbitos , un lenguaje puede volverse en una herramienta especializada para un tipo de problemas. Los conceptos que se usan en los compiladores de hoy en día, (como Ruby, C#), son conceptos que para construir tomaron demasiado ,tiempo. Conocer las bases del Desarrollo de compilación sirve para mejorar el uso de los lenguajes de programación actuales.

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

(Firma)

b) Descripción del Lenguaje

## b.1) Nombre del Lenguaje

Tuby

## b.2) Principales características del lenguaje

Tuby es un lenguaje orientado a objetos basado en el idioma español con propósito de ser utilizado como lenguaje Orientado a Objetos para poder dar una introducción al área de programación de alto nivel a personas con poco entendimiento del idioma inglés. El idioma deberá poder utilizar ciclos, condiciones, funciones y clases.

## b.3) Posibles errores de compilación/ejecución

#### Errores en compilacion

* No se cuenta con Python
* No se cuenta con la herramienta PLY
* No se cuentan archivos existentes con el nombre que se proporciono.

#### Errores en ejecución

* Errores de sintaxis, semántica y lógica que el compilador haya detectado.

c) Descripción del compilador

## c.1) Equipo de computo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto

El compilador se desarrolló en 2 equipos de cómputo Windows 10, con lenguaje Python y utilizando PLY como utilería y corriendo con el IDE Pycharm.

## c.2) Descripción de Análisis de Léxico.

#### Tokens asociados al lenguaje de programación

'entero': 'KEYWORD\_TYPE\_ENTERO',

'real': 'KEYWORD\_TYPE\_REAL',

'booleano': 'KEYWORD\_TYPE\_BOOLEANO',

'si': 'KEYWORD\_SI',

'sino': 'KEYWORD\_SINO',

'mientras': 'KEYWORD\_MIENTRAS',

'clase': 'KEYWORD\_CLASE',

'principal': 'KEYWORD\_PRINCIPAL',

'caracter': 'KEYWORD\_TYPE\_CARACTERES',

'entrada': 'KEYWORD\_ENTRADA',

'salida': 'KEYWORD\_SALIDA',

'funcion': 'KEYWORD\_FUNCION',

'nulo': 'KEYWORD\_NULO',

'retorno': 'KEYWORD\_RETORNO',

'verdadero': 'KEYWORD\_VERDADERO',

'falso': 'KEYWORD\_FALSO'

t\_SEMICOLON = r'\;'

t\_PUNTO = r'[\.]'

t\_COMMA = r'[\,]'

t\_COLON = r'\:'

t\_BRACKET\_IZQ = r'\{'

t\_BRACKET\_DER = r'\}'

t\_PARENTESIS\_IZQ = r'\('

t\_PARENTESIS\_DER = r'\)'

t\_CORCHETE\_IZQ = r'\['

t\_CORCHETE\_DER = r'\]'

t\_INTER\_IZQ = r'\¿'

t\_INTER\_DER = r'\?'

t\_OPERADOR\_IGUAL = r'\='

t\_OPERADOR\_COMPARATIVO = r'[>]|[<]'

t\_OPERADOR\_AND\_OR = r'&&|\|\|'

t\_EXP\_OPERADOR = r'\+|\-'

t\_TERM\_OPERADOR = r'\\*|\/'

t\_ignore = ' \t\n\r'

## c.3) Descripción de Análisis de Sintaxis

def t\_CONST\_NUMERO\_REAL(t):

r'[0-9]+\.[0-9]+'

t.value = float(t.value)

return t

def t\_CONST\_NUMERO\_ENT(t):

r'\d+'

t.value = int(t.value)

return t

def t\_IDENTIFICADOR(t):

r'[a-z\_][a-zA-Z0-9\_]\*'

if t.value in reserved:

t.type = reserved[t.value]

return t

def t\_IDENTIFICADOR\_CLASE(t):

r'[A-Z][a-zA-Z0-9\_]\*'

if t.value in reserved:

t.type = reserved[t.value]

return t

def t\_CONST\_BOOLEANO(t):

r'[KEYWORD\_VERDADERO|KEYWORD\_FALSO]'

return t

t\_CONST\_CARACTERES = r'\"[A-Za-z0-9\_\(\)\{\}\[\]\<\>\!\ ]\*\"'

def p\_empty(p):

'empty :'

pass

def p\_error(t):

print("Error de sintaxis en '%s'" % t.value)

def p\_Programa(t):

'''

Programa : Goto\_Principal ProgramaA FuncionPrincipal

'''

def p\_Goto\_Principal(p):

'''

Goto\_Principal :

'''

def p\_ProgramaA(t):

'''

ProgramaA : Declaracion ProgramaA

| Funcion ProgramaA

| Clase ProgramaA

| empty

'''

def p\_FuncionPrincipal(t):

'''

FuncionPrincipal : PrincipalAux INTER\_IZQ INTER\_DER Bloque FinBloquePrincipal

'''

def p\_PrincipalAux(t):

'''

PrincipalAux : KEYWORD\_PRINCIPAL

'''

def p\_FinBloquePrincipal(t):

'''

FinBloquePrincipal :

'''

def p\_Tipo(t):

'''Tipo : KEYWORD\_TYPE\_ENTERO

| KEYWORD\_TYPE\_REAL

| KEYWORD\_TYPE\_BOOLEANO

| KEYWORD\_TYPE\_CARACTERES

| IDENTIFICADOR\_CLASE\_AUX

'''

def p\_IDENTIFICADOR\_CLASE\_AUX(t):

'''

IDENTIFICADOR\_CLASE\_AUX : IDENTIFICADOR\_CLASE

'''

def p\_Asignacion(t):

''' Asignacion : IGUALSIM Expresion SEMICOLON

'''

def p\_IGUALSIM(t):

'''

IGUALSIM : OPERADOR\_IGUAL

'''

def p\_AsignaAux(t):

'''

AsignaAux : IDENTIFICADOR

'''

def p\_AsignaClass(t):

'''

AsignaClass : AsignaA

| PuntoAux AsignaAux AsignaA

'''

def p\_AsignaA(t):

'''

AsignaA : CORCHETE\_IZQ Expresion CORCHETE\_DER AsignaB

| empty

'''

def p\_AsignaB(t):

'''

AsignaB : CORCHETE\_IZQ Expresion CORCHETE\_DER

'''

def p\_Funcion(t):

'''

Funcion : FuncionAux INTER\_IZQ FuncionA INTER\_DER Bloque Fin\_Bloque

'''

def p\_FuncionAux(t):

'''

FuncionAux : KEYWORD\_FUNCION Tipo IDENTIFICADOR

'''

def p\_Fin\_Bloque(t):

'''

Fin\_Bloque :

'''

#algo similar a declaraA

def p\_FuncionA(t):

'''

FuncionA : DeclaraBase FuncionB

| empty

'''

def p\_FuncionB(t):

'''

FuncionB : COMMA FuncionA

| empty

'''

def p\_Parametro(t):

'''

Parametro : Tipo IDENTIFICADOR

'''

def p\_Bloque(t):

'''

Bloque : BRACKET\_IZQ BloqueA BRACKET\_DER

'''

def p\_BloqueA(t):

'''

BloqueA : Declaracion BloqueB

| DecOAss BloqueB

| Ciclo BloqueB

| Condicion BloqueB

| Entrada BloqueB

| Salida BloqueB

| Retorno Expresion SEMICOLON FinRetorno

'''

def p\_Retorno(t):

'''

Retorno : KEYWORD\_RETORNO

'''

def p\_FinRetorno(t):

'''

FinRetorno :

'''

def p\_DecOAss(t):

'''

DecOAss : AsignaAux AsignaClass DecOAssA

'''

def p\_DecOAssA(t):

'''

DecOAssA : LlamadaFuncion SEMICOLON

| Asignacion

'''

def p\_BloqueB(t):

'''

BloqueB : BloqueA

| empty

'''

def p\_Clase(t):

'''

Clase : ClaseAux Bloque\_Clase

'''

def p\_ClaseAux(t):

'''

ClaseAux : KEYWORD\_CLASE IDENTIFICADOR\_CLASE ClaseA

'''

def p\_ClaseA(t):

'''

ClaseA : COLON IDENTIFICADOR\_CLASE

| empty

'''

def p\_Bloque\_Clase(t):

'''

Bloque\_Clase : BRACKET\_IZQ Bloque\_ClaseA BRACKET\_DER SEMICOLON Fin\_Bloque\_Clase

'''

def p\_Fin\_Bloque\_Clase(t):

'''

Fin\_Bloque\_Clase :

'''

def p\_Bloque\_ClaseA(t):

'''

Bloque\_ClaseA : Bloque\_ClaseB Bloque\_ClaseC

'''

def p\_Bloque\_ClaseB(t):

'''

Bloque\_ClaseB : Declaracion Bloque\_ClaseB

| empty

'''

def p\_Bloque\_ClaseC(t):

'''

Bloque\_ClaseC : Funcion Bloque\_ClaseC

| empty

'''

def p\_Ciclo(t):

'''

Ciclo : CicloAux PARENTESIS\_IZQ Expresion CicloCheck Bloque

'''

def p\_CicloAux(t):

'''

CicloAux : KEYWORD\_MIENTRAS

'''

def p\_CicloCheck(t):

'''

CicloCheck : PARENTESIS\_DER

'''

def p\_Entrada(t):

'''

Entrada : KEYWORD\_ENTRADA IDENTIFICADOR SEMICOLON

'''

def p\_Salida(t):

'''

Salida : Salida\_Key\_Aux Expresion SEMICOLON Salida\_fin

'''

def p\_Salida\_Key\_Aux(t):

'''

Salida\_Key\_Aux : KEYWORD\_SALIDA

'''

def p\_Salida\_fin(t):

'''

Salida\_fin :

'''

def p\_Condicion(t):

'''

Condicion : CondicionAux PARENTESIS\_IZQ Expresion CondicionCheck Bloque TerminaCondicion CondicionA

'''

def p\_CondicionAux(t):

'''

CondicionAux : KEYWORD\_SI

'''

def p\_CondicionCheck(t):

'''

CondicionCheck : PARENTESIS\_DER

'''

def p\_TerminaCondicion(t):

'''

TerminaCondicion :

'''

def p\_CondicionA(t):

'''

CondicionA : SinoAux Bloque SinoBloqueFin

| empty

'''

def p\_SinoAux(t):

'''

SinoAux : KEYWORD\_SINO

'''

def p\_SinoBloqueFin(t):

'''

SinoBloqueFin :

'''

def p\_Expresion(t):

'''

Expresion : Expresion ExpresionA

| Expres

'''

def p\_ExpressionA(t):

'''

ExpresionA : ExpresionAux Expres

'''

def p\_ExpresionAux(t):

'''

ExpresionAux : OPERADOR\_AND\_OR

'''

def p\_Expres(t):

'''

Expres : Expres ExpresA

| Exp

'''

def p\_ExpresA(t):

'''

ExpresA : ExpresAux Exp

'''

def p\_ExpresAux(t):

'''

ExpresAux : OPERADOR\_COMPARATIVO

'''

def p\_Exp(t):

'''

Exp : Exp ExpA

| Termino

'''

def p\_ExpA(t):

'''

ExpA : ExpAux Termino

'''

def p\_ExpAux(t):

'''

ExpAux : EXP\_OPERADOR

'''

def p\_Termino(t):

'''

Termino : Termino TerminoA

| Factor

'''

def p\_TerminoA(t):

'''

TerminoA : TerminoAux Factor

'''

def p\_TerminoAux(t):

'''

TerminoAux : TERM\_OPERADOR

'''

def p\_Factor(t):

'''

Factor : ValorSalida

| ParentesisInit Expresion ParentesisFin

'''

def p\_ParentesisInit(t):

'''

ParentesisInit : PARENTESIS\_IZQ

'''

def p\_ParentesisFin(t):

'''

ParentesisFin : PARENTESIS\_DER

'''

def p\_LlamadaFuncion(t):

'''

LlamadaFuncion : INTER\_IZQ LlamadaFuncionA INTER\_DER FinalLlamada

'''

def p\_LlamadaFuncionA(t):

'''

LlamadaFuncionA : Expresion CorreExpresion LlamadaFuncionB

| empty

'''

def p\_CorreExpresion(t):

'''

CorreExpresion :

'''

def p\_LlamadaFuncionB(t):

'''

LlamadaFuncionB : COMMA LlamadaFuncionA

| empty

'''

def p\_FinalLlamada(t):

'''

FinalLlamada :

'''

def p\_DeclaraBase(t):

'''

DeclaraBase : Parametro DeclaraA

'''

def p\_Declaracion(t):

'''

Declaracion : DeclaraBase SEMICOLON

'''

def p\_DeclaraA(t):

'''

DeclaraA : CORCHETE\_IZQ CONST\_NUMERO\_ENT CORCHETE\_DER DeclaraB

| empty

'''

def p\_DeclaraB(t):

'''

DeclaraB : CORCHETE\_IZQ CONST\_NUMERO\_ENT CORCHETE\_DER

| empty

'''

def p\_ValorSalida(t):

'''

ValorSalida : NumeroEntero

| Caracter

| NumeroReal

| Booleano

| KEYWORD\_NULO

| LlamadaIDs

'''

def p\_LlamadaIDs(t):

'''

LlamadaIDs : LlamadaIDsAux LlamadaIDsA

'''

def p\_LlamadaIDsAux(t):

'''

LlamadaIDsAux : IDENTIFICADOR

'''

def p\_LlamadaIDsA(t):

'''

LlamadaIDsA : Terminal

| LlamadaFuncion

'''

def p\_NumeroEntero(t):

'''

NumeroEntero : CONST\_NUMERO\_ENT

'''

def p\_Caracter(t):

'''

Caracter : CONST\_CARACTERES

'''

def p\_NumeroReal(t):

'''

NumeroReal : CONST\_NUMERO\_REAL

'''

def p\_Booleano(t):

'''

Booleano : CONST\_BOOLEANO

'''

def p\_Terminal(t):

'''

Terminal : AsignaClass

'''

def p\_ValorSalidaB(t):

'''

ValorSalidaB : PuntoAux IdentificadorAux ValorSalidaC

| empty

'''

def p\_IdentificadorAux(t):

'''

IdentificadorAux : IDENTIFICADOR

'''

def p\_PuntoAux(t):

'''

PuntoAux : PUNTO

'''

def p\_ValorSalidaC(t):

'''

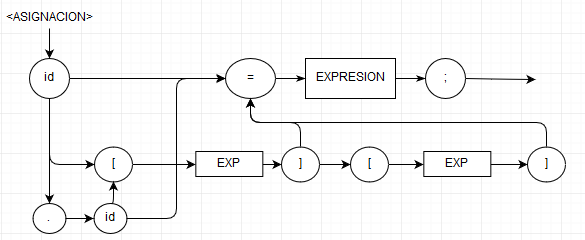
ValorSalidaC : INTER\_IZQ LlamadaFuncionA INTER\_DER

| AsignaA ValorSalidaB

'''

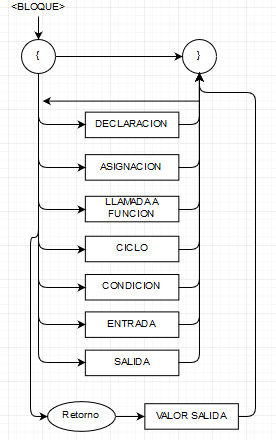
## c.4) Descripción de Generación de Código Intermedio

#### Asignación



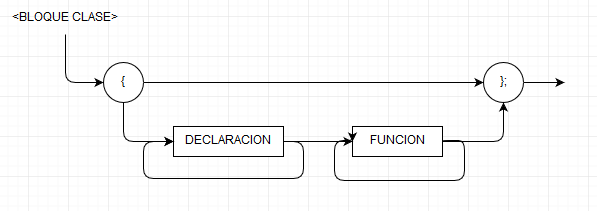
Nuestro diagrama de sintaxis de Asignación está desarrollado con la opción de proveer múltiples ids para poder hacer referencia a las funciones dentro de clases dentro del lenguaje. También cuenta con la opción de introducir el valor de matrices o arreglos.

#### Bloque



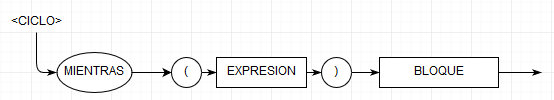
Bloque nos permite introducir información a correr dentro de cada función nueva que se encuentra encapsulado entre corchetes.

#### Bloque-Clase



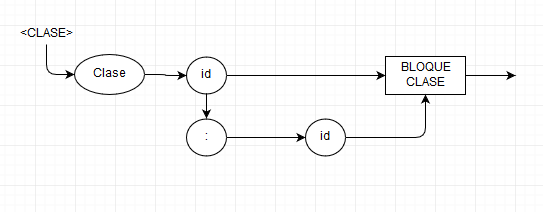
Bloque-Clase es un bloque especial diseñado para tener uso de funciones de clase con sus valores específicos.

#### Ciclo



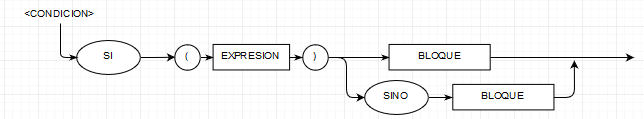
Ciclo es nuestra función while, se le va a proveer una expresión dentro de paréntesis para poder decidir cuando se sale del ciclo.

#### Clase



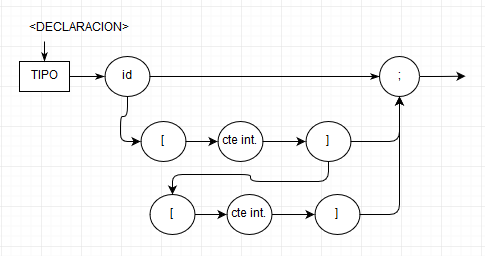
Definición de clase, los 2 puntos se utilizan al momento de realizar herencia entre las clases.

#### Condición



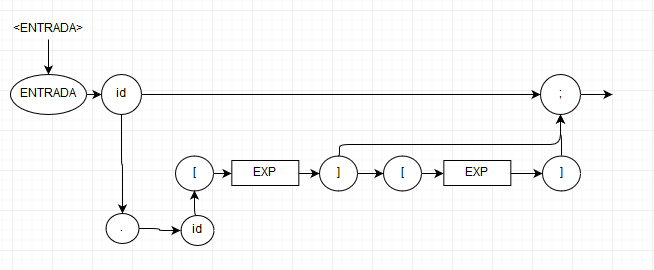
Condicion es nuestra sintaxis de if y else de nuestro lenguaje. Si se cumple con la condicion de la primer expresion, se corre el primer bloque, si no, corre el bloque de sino o simplemente se salta ambos bloques si es que no se encuentra una condicion Sino.

#### Declaracion



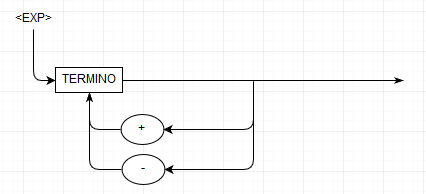
Declaración se utiliza para definir una nueva variable, si es arreglo o matriz se introducen la cantidad de brackets necesarios.

#### Entrada



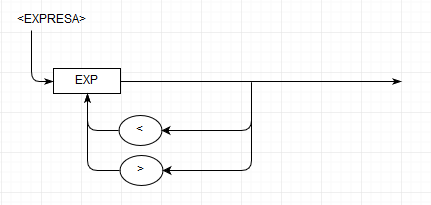
Valor de entrada fuera del programa. Se le pide al usuario teclear lo que se desea introducir dentro del programa.

#### Exp



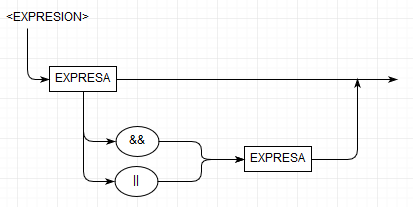
Parte de del orden de Expresión en el que se ingresan los operadores de suma o resta.

#### Expresa



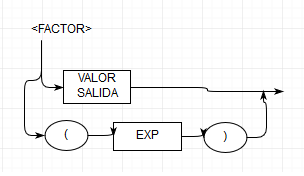
Parte del orden de sintaxis de Expresión encargada de proveer los operadores comparativos de menor que y mayor que.

#### Expresión



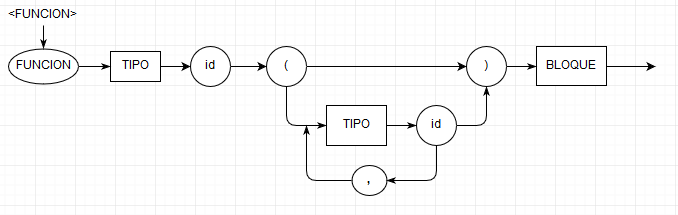
Expresión se dedica proveer valores numéricos o booleanos a través de un orden sintáctico de las operaciones a ejecutar. Empieza con los operadores de AND y OR.

#### Factor



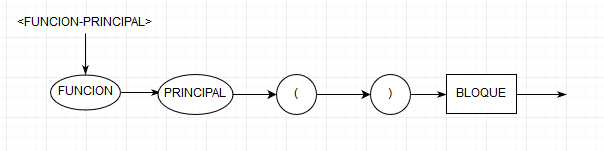
Factor es la última parte de Expresión, en el cual se proveen los valores que serán distribuidos a la expresión por realizar o introduce jerarquía de operaciones dentro de la expresión.

#### Funcion



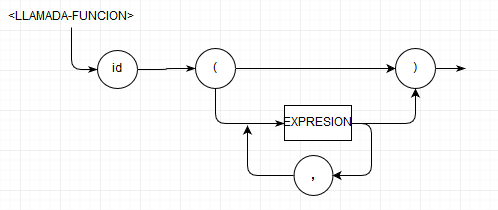
Nuestro manejo de sintaxis de Funciones, se pueden proveer múltiples parámetros, y al terminar de proveerlos se inicia el bloque en el que se realizarán las acciones.

#### Funcion-Principal



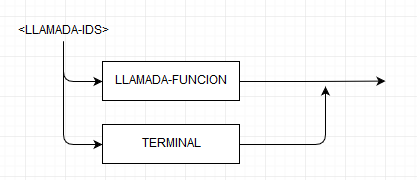
Función que utilizamos como main para nuestro programa, es donde se inicia la ejecución de los cuádruplos y de allí se empiezan a llamar los otros elementos dentro del programa.

#### Llamada-Función



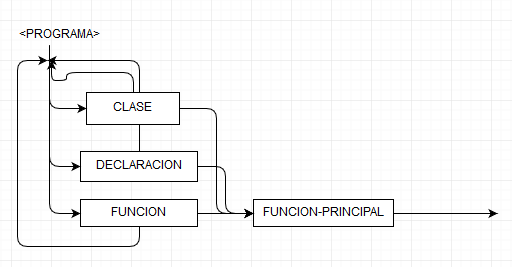
Llamada-Función es lo que utilizamos para hacer una llamada a los datos que una función nos va a proporcionar dentro de nuestro programa.

#### Llamada-IDs



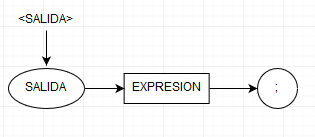
Esta tabla nos ayuda a ordenar la llamada entre Llamada-Función y valores Terminales que podríamos llamar desde Valor Salida

#### Programa



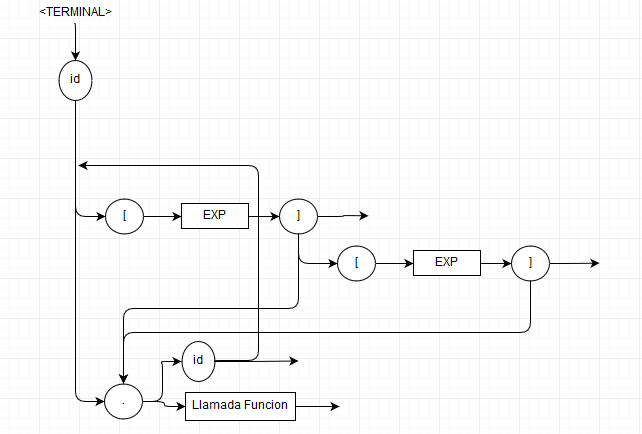
Es la llamada a nuestro programa a correr en nuestro compilador. Dentro de el se pueden realizar declaraciones globales, clases o funciones antes de iniciar la función principal.

#### Salida



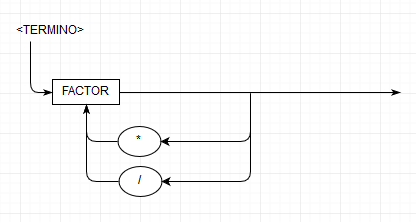
Es el valor a desplegar en pantalla para que el usuario pueda ver resultados de su programa.

#### Terminal



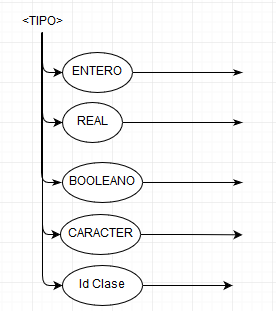
Valores terminales del programa, es aquí donde se hace referencia a matrices o llamadas a función de clases como valores de salida.

#### Termino



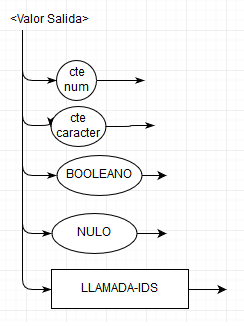
Término es el otra parte de Expresión en la que se registran los operadores de multiplicación y división en el orden de cuádruplos.

#### Tipo



Definición del tipo de nuestras variables, se le regresa el valor cada vez que se hace referencia dentro del programa.

#### Valor Salida



Valores de salida que se introducirán dentro de las expresiones del programa.

## c.5) Descripción del proceso de Administración de memoria

Como se estuvo trabajando en un lenguaje orientado a objetos , la memoria no podía ser estática, debía de tener una manera de pedir más recursos , identificar las variables del objeto fácilmente. Con lo tal se genero el siguiente sistema. La memoria física se compone de una lista original de 4 monolitos , Global,Local,Constante y Temporal. Las cuales pertenecían a los índices 1,2 3,4 respectivamente. Dentro de cada uno de estos monolitos , el monolito se dividió en 4, del 1 al 2500 para booleanos, 2501 al 5000 en enteros , 5001 al 7500 en reales y los restantes 2500 en caracteres. Cuando se genera una nueva variable se instancia en uno de estos. (dependiendo de su tipo). Ahora Si se genera una variable de clase, agarra la memoria auxiliar que se generó (Memoria Clase). Que empieza del 500 0000. Se instancia una una nueva lista de memoria, se agrega a las listas de memoria y cada que se instancia una clase de memoria sigue los siguientes pasos:

1 crea nueva lista de memoria de Clase

2 crea nueva lista de memoria para funciones de clase

3 enviarlo a la lista de memoria

4 instanciar una variable (del nombre de la clase), dentro de la Memoria clase

Así , simplemente viendo el numero puedes saber que tipo de variables son

Del 0 al 10000 son variables globales del 10001 al 20000 locales 20001 al 30000 son constantes y del 30001 al 40000 son constantes.

Variables que tengan identificador mayor a 400000 pertenecer a variables de las clases.

d) Descripción de la Máquina Virtual

## d.1) Equipo de computo, lenguaje y utilerías especiales utilizadas.

Se utilizo el mismo equipo de computo que se habia utilizando para compilar el programa. Se utilizo un equipo Windows 10 utilizando Pycharm para correr el codigo scanner.py y se utilizo la utileria Ply para poder correrlo.

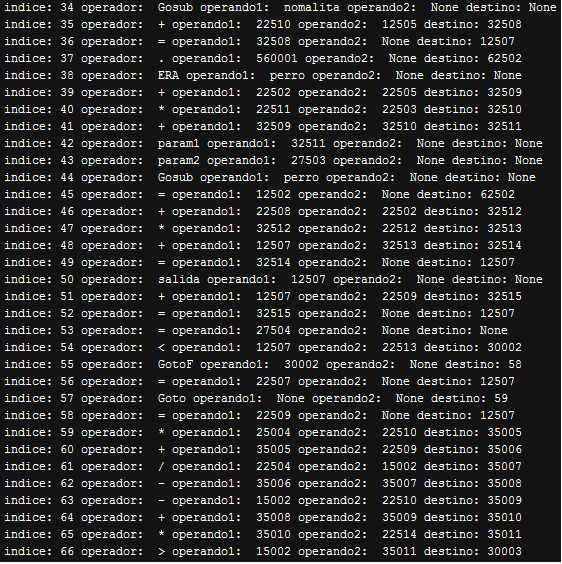
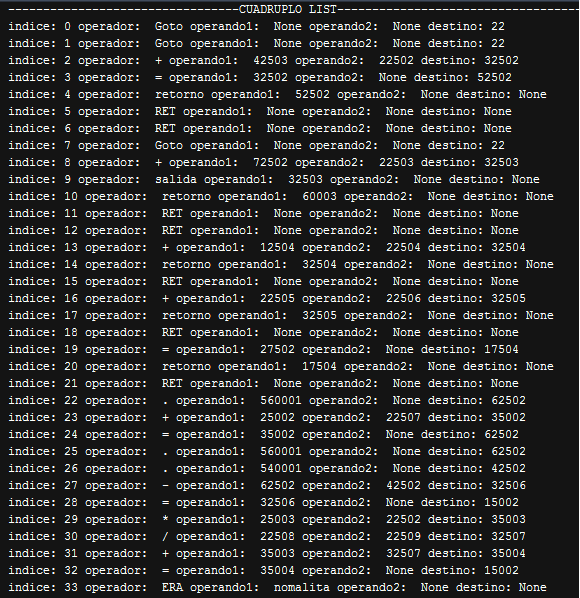
## d.2) Descripción detallada del proceso de Administración de Memoria en ejecución.

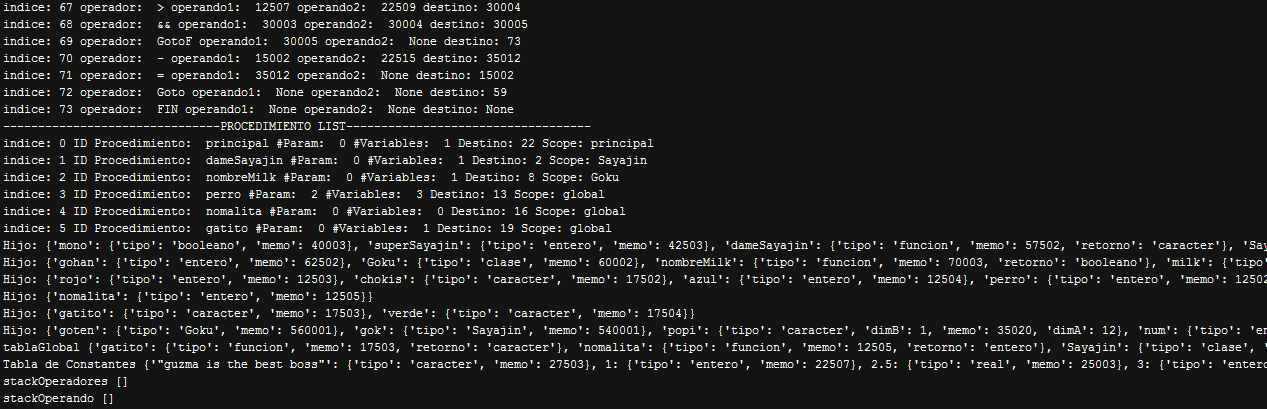
El proceso de administración de memoria en ejecución trabajo utilizando las tablas de cuadruplos para saber a dónde dirigirse en cuestión de ejecución de cada instrucción que se le manda al compilador. La máquina lee el operador por cuádruplo y realiza una acción de acuerdo al operador. Recibe las direcciones de memoria real y los traduce a valores de memoria virtual para así transferir datos entre espacios de memoria.

e) Pruebas del funcionamiento del lenguaje

## e.1) Pruebas que comprueben el funcionamiento del lenguaje

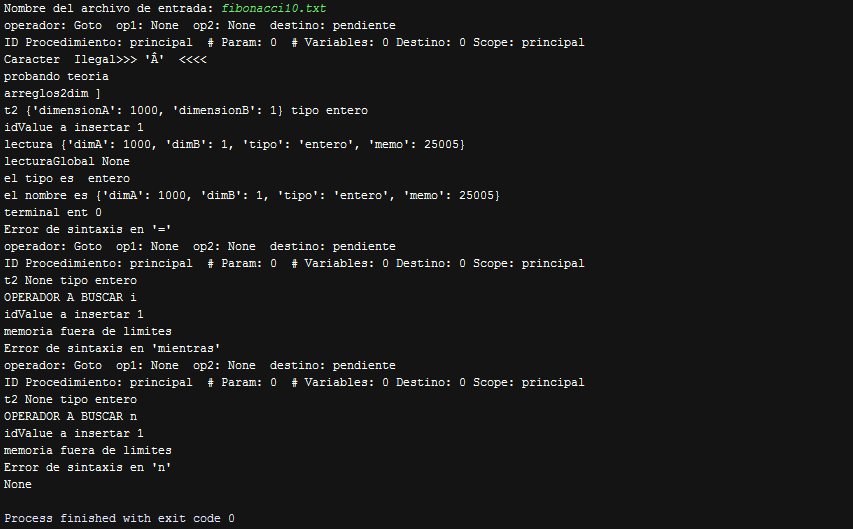
#### Prueba con archivo propio hello.txt





Se pudieron realizar los cuadruplos, las tablas de procesos, espacios de memoria y lista de simbolos, pero no se pudo realizar completamente los valores de retorno a consola.

#### Prueba Fibonacci con valor n=10



No se pudo entrar a los valores de memoria de Arreglo.

f) Listados Perfectamente Documentados del Proyecto

## Scanner.py

#diagrama de sintaxis Inicial del pograma , Funcion principal

def p\_Programa(t):

'''

Programa : Goto\_Principal ProgramaA FuncionPrincipal

'''

print('La sintaxis del programa paso')

# print ('Global scope symbols:')

global tablaSimbolosActual,cuadruploList,stackOperador, procedimientoList, stackOperando, procedimientoList, maquinaVirtual

print('global scope symbols:', tablaSimbolosActual.simbolos)

cuadruploList.normalCuad('FIN',None,None,None)

print("---------------------------------CUADRUPLO LIST-------------------------------------")

cuadruploList.imprimir()

print("-------------------------------PROCEDIMIENTO LIST-----------------------------------")

procedimientoList.imprimir()

tablaGlobal.imprimir()

tablaConstantes.imprimir()

maquinaVirtual.setCuad(cuadruploList.getCuadruplos())

maquinaVirtual.setProc(procedimientoList.getProcedimientos())

maquinaVirtual.setSimbolos(tablaGlobal)

maquinaVirtual.setConstantes(tablaConstantes)

maquinaVirtual.setMemTemp(memoriaTemporal)

print('stackOperadores',stackOperador)

print('stackOperando', stackOperando)

maquinaVirtual.calculos()

#goto que general el cuadruplo de la funcion principal , hacer uqe sea efectivo.

def p\_Goto\_Principal(p):

'''

Goto\_Principal :

'''

global cuadruploList,pilaSaltos, procedimientoList

cuadruploList.normalCuad('Goto',None,None, 'pendiente')

procedimientoList.normalLista("principal", 0, 0, 0, "principal")

#diagrama de sintaxis de como funciona el programa , puedes ahcer declaraciones, Funciones o clases antes de entrar al main

def p\_ProgramaA(t):

'''

ProgramaA : Declaracion ProgramaA

| Funcion ProgramaA

| Clase ProgramaA

| empty

'''

#funcion que corre el programa principal , necesaria para correr nuestro lenguaje

def p\_FuncionPrincipal(t):

'''

FuncionPrincipal : PrincipalAux INTER\_IZQ INTER\_DER Bloque FinBloquePrincipal

'''

#diagrama para Identificar cuando entra a la funcion principal , generar tabla de simbolos

def p\_PrincipalAux(t):

'''

PrincipalAux : KEYWORD\_PRINCIPAL

'''

global tablaSimbolosActual, claseJumps,listaMemorias, procedimientoList, varLocal, proScope

proScope = t[1]

mem = listaMemorias[1].insertaEntero()

tablaSimbolosActual.insertarFuncion(t[1], 'entero', mem)

tablaM = TablaSimbolos()

tablaM.id = mem

tablaM.agregarPadre(tablaSimbolosActual)

tablaSimbolosActual.agregarHijo(tablaM)

tablaSimbolosActual = tablaM

cuadruploList.updateCuad(0, "Goto", None, None, cuadruploList.CuadSize())

procedimientoList.updateLista(0, "principal", 0, varLocal, cuadruploList.CuadSize(), proScope)

for x in claseJumps:

cuadruploList.updateCuad(x-1, "Goto", None, None, cuadruploList.CuadSize())

#Regresa a la tabla de simbolos actual

def p\_FinBloquePrincipal(t):

'''

FinBloquePrincipal :

'''

global tablaSimbolosActual, cuadruploList, procedimientoList, proScope

tablaSimbolosActual = tablaSimbolosActual.padre

proScope = "global"

#Sirve para identificar y devolver el tipo

def p\_Tipo(t):

'''Tipo : KEYWORD\_TYPE\_ENTERO

| KEYWORD\_TYPE\_REAL

| KEYWORD\_TYPE\_BOOLEANO

| KEYWORD\_TYPE\_CARACTERES

| IDENTIFICADOR\_CLASE\_AUX

'''

t[0] = t[1]

#sirve para identificar si una clase existe.

def p\_IDENTIFICADOR\_CLASE\_AUX(t):

'''

IDENTIFICADOR\_CLASE\_AUX : IDENTIFICADOR\_CLASE

'''

existe = tablaGlobal.buscar(t[1])

if (existe is None):

print("Tipo no existente, clase no declarada")

print("verifica generar la clase ANTES de mandarla a llamar")

raise SyntaxError

else:

t[0] = t[1]

#diagrama de sintaxis usado para asignar valores a una variable.

def p\_Asignacion(t):

''' Asignacion : IGUALSIM Expresion SEMICOLON

'''

# parte de heacuadruplo para expresion

global stackOperador,stackOperando,cuadruploList

op=stackOperador.pop()

operando = stackOperando.pop()

destino = stackOperando.pop()

cuadruploList.AssignCuad(op,operando,destino)

#sirve para meter el operador igual en el stack de operadores.

def p\_IGUALSIM(t):

'''

IGUALSIM : OPERADOR\_IGUAL

'''

global stackOperador

stackOperador.append(t[1])

#identificador de llamadas, o asignaciones.

def p\_AsignaAux(t):

'''

AsignaAux : IDENTIFICADOR

'''

global tablaSimbolosActual, tablaGlobal, buscadorClase, stackOperando, auxstackParam, procedimientoList,cuadruploList

existe = tablaSimbolosActual.buscar(t[1])

existeglobal = tablaGlobal.buscar(t[1])

if (buscadorClase is None):

if (existe is None):

print("variable no existe en este punto", buscadorClase)

elif (existe['tipo'] == 'real' or existe['tipo'] == 'booleano' or existe['tipo'] == 'caracter' or existe['tipo'] == 'entero'):

stackOperando.append(existe['memo'])

elif (not (existe['tipo'] == 'real' or existe['tipo'] == 'booleano' or existe['tipo'] == 'caracter' or existe['tipo'] == 'entero')):

if (existe['tipo'] == 'funcion'):

print("no puedes hacer asignacion con funcion")

else:

buscadorClase = tablaGlobal.buscarHijos(existe['tipo'])

if (not (buscadorClase is None)):

stackOperando.append(existe['memo'])

else:

print("clase no encontrada");

raise SyntaxError

if(not(existeglobal is None)):

if (existeglobal['tipo'] == 'funcion'):

print("ZORDON")

auxstackParam.append(t[1])

auxstackParam.append(procedimientoList.buscar(t[1]))

if (auxstackParam[1][0] is not None):

auxstackParam[1].reverse()

print("SE TIENEN LOS PARAMETROS EN LA FUNCION: ", auxstackParam)

else:

print('LectoraClase',buscadorClase.simbolos)

existe = buscadorClase.buscar(t[1])

print("buscar dentro de clase", existe)

if (existe is None):

print("Clase No encontrada")

raise SyntaxError

elif (existe['tipo'] == 'real' or existe['tipo'] == 'booleano' or existe['tipo'] == 'caracter' or existe['tipo'] == 'entero'):

print("aqui meter en vector que es una variable de tipo: ", existe)

if(stackOperador[len(stackOperador)-1]== '.'):

op = stackOperador.pop()

op1 = stackOperando.pop()

cuadruploList.normalCuad(op,op1,None,existe['memo'])

stackOperando.append(existe['memo'])

buscadorClase = None

elif (existe['tipo'] == 'funcion'):

buscadorClase = None

else:

buscadorClase = tablaGlobal.buscarHijos(existe['tipo'])

if(buscadorClase is None):

print("Error de sintaxis clase no existe")

else:

buscadorClase.id

print("marcar error")

#diagrama utilizado para encontrar las llamadas a funcion de una clase, o accesso de varaibles

def p\_AsignaClass(t):

'''

AsignaClass : AsignaA

| PuntoAux AsignaAux AsignaA

'''

#diagrama usado para instanciar vectores.

def p\_AsignaA(t):

'''

AsignaA : CORCHETE\_IZQ Expresion CORCHETE\_DER AsignaB

| empty

'''

#diagrama usado para instanciar matrices

def p\_AsignaB(t):

'''

AsignaB : CORCHETE\_IZQ Expresion CORCHETE\_DER

| empty

'''

#diagrama para generar los parametros dentro de una funcion.

def p\_Funcion(t):

'''

Funcion : FuncionAux INTER\_IZQ FuncionA INTER\_DER Bloque Fin\_Bloque

'''

#diagrama para generar e instanciar funciones

def p\_FuncionAux(t):

'''

FuncionAux : KEYWORD\_FUNCION Tipo IDENTIFICADOR

'''

global tablaSimbolosActual, stackParam, cuadruploList, cuadruActual, varLocal,tablaGlobal,listaMemorias

stackParam = []

existe = tablaSimbolosActual.buscar(t[3])

if (existe is None):

tipoID = t[2]

memID = 0

if(not (tipoID == 'entero' or tipoID =='booleano' or tipoID =='caracter' or tipoID =='real')):

idValue = tablaSimbolosActual.id+10000

idValue =int(idValue/10000)

existe = tablaGlobal.buscarHijos(t[2])

if(existe is None):

print("Tipo de clase no ha sido declarada")

else:

memID = memoriaIDClases.insertarClase(idValue+10000)

########################################################################ACABAR CLASE

else:

idValue = int(tablaSimbolosActual.id/10000)

idValue =idValue + 1

if(tipoID =='entero'):

memID = listaMemorias[idValue].insertaEntero()

elif(tipoID =='booleano'):

memID = listaMemorias[idValue].insertaBooleano()

elif(tipoID =='caracter'):

memID = listaMemorias[idValue].insertaCaracter()

elif(tipoID =='real'):

memID = listaMemorias[idValue].insertaReal()

tablaSimbolosActual.insertarFuncion(t[3], t[2], memID) # guarda que es Tipo funcion en la tabla de simbolos

tablaF = TablaSimbolos()

tablaF.id = memID

tablaF.insertar(t[3], t[2],memID)

tablaSimbolosActual.agregarHijo(tablaF)

tablaF.agregarPadre(tablaSimbolosActual) #

tablaSimbolosActual = tablaF

stackParam.append(t[3])

cuadruActual = cuadruploList.CuadSize()

varLocal = 0

else:

print("Funcion previamente declarada")

raise SyntaxError

#diagrama para salir de un bloque , preferiblemente de funciones

def p\_Fin\_Bloque(t):

'''

Fin\_Bloque :

'''

global tablaSimbolosActual,cuadruploList, stackParam, procedimientoList, cuadruActual, varLocal, proScope

tablaSimbolosActual = tablaSimbolosActual.padre

cuadruploList.normalCuad('RET',None,None,None)

functId = stackParam.pop(0)

procedimientoList.meteParametros(functId, stackParam)

if (stackParam[0] is None):

cantParam = 0;

else:

cantParam = len(stackParam)

procedimientoList.normalLista(functId, cantParam, varLocal, cuadruActual, proScope)

#algo similar a declaraA

def p\_FuncionA(t):

'''

FuncionA : DeclaraBase FuncionB

| empty

'''

global stackParam

stackParam.append(t[1])

#instancias multiples parametros de variables

def p\_FuncionB(t):

'''

FuncionB : COMMA FuncionA

| empty

'''

#tipo devuelve el tipo y junto con identificador , instancia un objeto en la tabla de variables

def p\_Parametro(t):

'''

Parametro : Tipo IDENTIFICADOR

'''

global tablaSimbolosActual,tablaGlobal,memoriaIDClases,stackOperando

existe = tablaSimbolosActual.buscar(t[2])

if (existe is None):

existe = tablaSimbolosActual.padre

if(not(tablaSimbolosActual.padre is None)):

existe = tablaSimbolosActual.padre.buscar(t[2])

if(existe is None):

stackOperando.append(t[2])

t[0] = t[1]

else:

stackOperando.append(t[2])

t[0] = t[1]

else:

print("variable previamente declarada")

raise SyntaxError

#inicio de un bloque

def p\_Bloque(t):

'''

Bloque : BRACKET\_IZQ BloqueA BRACKET\_DER

'''

#cosas que puede hacer dentro de un bloque

#ciclos, Asignacoin, llamadas a funcion , condiciones, inputs de entrada, inputs de salida, y retorno

def p\_BloqueA(t):

'''

BloqueA : Declaracion BloqueB

| DecOAss BloqueB

| Ciclo BloqueB

| Condicion BloqueB

| Entrada BloqueB

| Salida BloqueB

| Retorno Expresion SEMICOLON FinRetorno

'''

# guarda el retorno en el stack de operadores

def p\_Retorno(t):

'''

Retorno : KEYWORD\_RETORNO

'''

global stackOperador

stackOperador.append(t[1])

#genera el cuadruplo corespondiente a retorno

def p\_FinRetorno(t):

'''

FinRetorno :

'''

global cuadruploList,stackOperando,stackOperador

op = stackOperador.pop()

op1 = stackOperando.pop()

cuadruploList.normalCuad(op,op1,None,None)

#diagrama para hacer o una llamada funcion o una asignacion

def p\_DecOAss(t):

'''

DecOAss : AsignaAux AsignaClass DecOAssA

'''

#diagrama complementario del pasado

def p\_DecOAssA(t):

'''

DecOAssA : LlamadaFuncion SEMICOLON

| Asignacion

'''

#diagrama para manejar la recursion de elemento sdentro de un bloque

def p\_BloqueB(t):

'''

BloqueB : BloqueA

| empty

'''

#diagrama para empezar a instanciar una clase

def p\_Clase(t):

'''

Clase : ClaseAux Bloque\_Clase

'''

#diagrama complementario del anterior

def p\_ClaseAux(t):

'''

ClaseAux : KEYWORD\_CLASE IDENTIFICADOR\_CLASE ClaseA

'''

global tablaSimbolosActual, llavetablaclase, cuadruploList, claseJumps,nuevasClases,listaMemorias, proScope

proScope = t[2]

existe = tablaSimbolosActual.buscar(t[2])

if (existe is None):

nuevaClaseG = MemoriaReal(nuevasClases)

nuevasClases = nuevasClases + 10000

nuevaClaseL = MemoriaReal(nuevasClases)

nuevasClases = nuevasClases + 10000

memo = nuevaClaseG.insertaBooleano()

listaMemorias.append(nuevaClaseG)

listaMemorias.append(nuevaClaseL)

memob = nuevaClaseL.insertaBooleano()

if (llavetablaclase is None):

tablaSimbolosActual.insertarClase(t[2], memo)

tablaC = TablaSimbolos()

tablaC.insertar(t[2], 'clase',memo)

tablaC.id = memo

tablaC.agregarPadre(tablaSimbolosActual)

tablaSimbolosActual.agregarHijo(tablaC)

tablaSimbolosActual = tablaC

else:

heredado = t[1] + "," + llavetablaclase

tablaSimbolosActual.insertarClase(t[2],memo ,llavetablaclase)

tablaC = TablaSimbolos()

tablaC.insertar(t[2], 'clase',memo)

tablaC.agregarPadre(tablaSimbolosActual)

tablaC.id = memo

tablaSimbolosActual.agregarHijo(tablaC)

tablaSimbolosActual = tablaC

llavetablaclase = ""

cuadruploList.normalCuad('Goto',None, None, 'pendienteClase')

claseJumps.append(cuadruploList.CuadSize())

else:

print("Clase ya existente");

raise SyntaxError

#{'tipo':'clase','memo': memID,'id':id,'herencia':herencia}

#Funcion para definir herencia

def p\_ClaseA(t):

'''

ClaseA : COLON IDENTIFICADOR\_CLASE

| empty

'''

global tablaSimbolosActual, llavetablaclase

if (len(t) == 3):

existe = tablaSimbolosActual.buscar(t[2])

if (existe is None):

print("Clase a heredar no existente")

else:

existe['herencia']

if (not(existe['herencia'] is None)):

print("Herencia maxima de 1 solo nivel");

raise SyntaxError

else:

llavetablaclase = t[2]

#como bloque , mas enfocado a las clases , con la diferencia uqe un bloque de clase termina en semicolon

def p\_Bloque\_Clase(t):

'''

Bloque\_Clase : BRACKET\_IZQ Bloque\_ClaseA BRACKET\_DER SEMICOLON Fin\_Bloque\_Clase

'''

print("haber cuando corriste");

#regresar a la tabla global despues de salir de clases.

def p\_Fin\_Bloque\_Clase(t):

'''

Fin\_Bloque\_Clase :

'''

global tablaSimbolosActual, cuadruploList, proScope

tablaSimbolosActual = tablaSimbolosActual.padre

cuadruploList.normalCuad('RET')

proScope = "global"

#funcion para manerar si asignar declaracion o funciones dentro de la clase objeto

def p\_Bloque\_ClaseA(t):

'''

Bloque\_ClaseA : Bloque\_ClaseB Bloque\_ClaseC

'''

#diagrama especifico para Declaracion

def p\_Bloque\_ClaseB(t):

'''

Bloque\_ClaseB : Declaracion Bloque\_ClaseB

| empty

'''

#diagrama especifico para clase

def p\_Bloque\_ClaseC(t):

'''

Bloque\_ClaseC : Funcion Bloque\_ClaseC

| empty

'''

#diagrama para las reglas de Ciclo

#ciclo check termina generando el cuadruplo de ciclo

def p\_Ciclo(t):

'''

Ciclo : CicloAux PARENTESIS\_IZQ Expresion CicloCheck Bloque

'''

global cuadruploList, indiceCondicion

Ciclodir, CicloCheck = t[1], t[4]

cuadruploList.SaltaCuad("Goto", Ciclodir)

cuadruploList.AgregarSalto(CicloCheck, indiceCondicion)

#mete al stack de operadores la keyword mientras comoGotoF

def p\_CicloAux(t):

'''

CicloAux : KEYWORD\_MIENTRAS

'''

global stackOperador, cuadruploList

stackOperador.append("GotoF")

t[0] = cuadruploList.CuadSize()

#termina el cuadruplo de ciclo

def p\_CicloCheck(t):

'''

CicloCheck : PARENTESIS\_DER

'''

global cuadruploList, stackOperador, indiceCondicion, stackOperando

op = stackOperador.pop()

indiceCondicion = stackOperando.pop()

t[0] = cuadruploList.SaltaCuad(op)

#diagrama para generar inputs de entrada

def p\_Entrada(t):

'''

Entrada : KEYWORD\_ENTRADA IDENTIFICADOR SEMICOLON

'''

#diagrama para generar salida de variables y expresiones a terminal

def p\_Salida(t):

'''

Salida : Salida\_Key\_Aux Expresion SEMICOLON Salida\_fin

'''

#Agrega Salida al stack de operadores

def p\_Salida\_Key\_Aux(t):

'''

Salida\_Key\_Aux : KEYWORD\_SALIDA

'''

global stackOperador

stackOperador.append(t[1])

#funcion que finaliza el cuadruplo

def p\_Salida\_fin(t):

'''

Salida\_fin :

'''

global cuadruploList,stackOperando,stackOperador

op = stackOperador.pop()

op1 = stackOperando.pop()

cuadruploList.normalCuad(op,op1)

#diagrama de condicion

def p\_Condicion(t):

'''

Condicion : CondicionAux PARENTESIS\_IZQ Expresion CondicionCheck Bloque TerminaCondicion CondicionA

'''

global cuadruploList, indiceCondicion, elseDir

termina = t[4]

cuadruploList.AgregarSalto(termina, indiceCondicion, elseDir)

#diagrama para meter al stack de operadores un gotof si se encuentra con un Si

def p\_CondicionAux(t):

'''

CondicionAux : KEYWORD\_SI

'''

global stackOperador

stackOperador.append("GotoF")

#actualiza el cuadruplo

def p\_CondicionCheck(t):

'''

CondicionCheck : PARENTESIS\_DER

'''

global cuadruploList, stackOperador, saltoCond, indiceCondicion, stackOperando

op = stackOperador.pop()

saltoCond = cuadruploList.SaltaCuad(op)

indiceCondicion = stackOperando.pop()

t[0] = saltoCond

#diagrama para cargar la instruccion Goto al terminar la condicion

def p\_TerminaCondicion(t):

'''

TerminaCondicion :

'''

global cuadruploList

cuadruploList.SaltaCuad("Goto")

#corre el bloque sino si es que no se cumple la condicion de si

def p\_CondicionA(t):

'''

CondicionA : SinoAux Bloque SinoBloqueFin

| empty

'''

#corre los los contadores de tamano de cuadruplo para saber donde ir al empezar sino

def p\_SinoAux(t):

'''

SinoAux : KEYWORD\_SINO

'''

global cuadruploList, elseDir

elseDir = cuadruploList.CuadSize()

#se actualiza la direccion de salto al terminar el bloque de sino

def p\_SinoBloqueFin(t):

'''

SinoBloqueFin :

'''

global cuadruploList

cuadruploList.AgregarSalto(elseDir-1, None, cuadruploList.CuadSize())

#cuadro principal de expresion

def p\_Expresion(t):

'''

Expresion : Expresion ExpresionA

| Expres

'''

#Llama auxiliar y la siguiente parte de la expresion

def p\_ExpressionA(t):

'''

ExpresionA : ExpresionAux Expres

'''

global stackOperador, stackOperando, cuadruploList, temporales, indicetemporales, checkSemantica,listaMemorias

top = stackOperador[len(stackOperador) - 1]

print("OPERADORES HASTA EL MOMENTO AND OR", stackOperador)

if (top == '&&' or top == '||'):

temporales[indicetemporales] = "temporalExpresion"

op = stackOperador.pop()

oper2 = stackOperando.pop()

oper1 = stackOperando.pop()

sem = checkSemantica.Semantica(op,oper1, oper2)

memID = 0

if(sem == 0):

memID = listaMemorias[3].insertaBooleano()

elif(sem == 1):

memID = listaMemorias[3].insertaEntero()

elif(sem == 2):

memID = listaMemorias[3].insertaReal()

elif(sem == 3):

memID = listaMemorias[3].insertaCaracter()

elif (sem == 5):

print("ERROR, los tipos de datos proveidos no son compatibles entre si.")

cuadruploList.normalCuad(op, oper1, oper2, memID)

#INSERTAR FUNCION DE CHEQUEO DE SEMANTICA, INSERTAR TEMPORALES ADECUADOS

stackOperando.append(memID)

#recibe los operadores And y Or

def p\_ExpresionAux(t):

'''

ExpresionAux : OPERADOR\_AND\_OR

'''

global stackOperador

stackOperador.append(t[1])

#siguiente parte de expresion, corre otras jerarquias de expresion

def p\_Expres(t):

'''

Expres : Expres ExpresA

| Exp

'''

#llama la auxiliar de expresion y un nivel mas abajo de expresion

def p\_ExpresA(t):

'''

ExpresA : ExpresAux Exp

'''

global stackOperador, stackOperando, cuadruploList, temporales, indicetemporales, checkSemantica,listaMemorias

top = stackOperador[len(stackOperador) - 1]

print("OPERADORES HASTA EL MOMENTO COMPARATIVO", stackOperador)

if (top == '<' or top == '>'):

temporales[indicetemporales] = "temporalExpres"

op = stackOperador.pop()

oper2 = stackOperando.pop()

oper1 = stackOperando.pop()

sem = checkSemantica.Semantica(op,oper1, oper2)

memID = 0

if(sem == 0):

memID = listaMemorias[3].insertaBooleano()

elif(sem == 1):

memID = listaMemorias[3].insertaEntero()

elif(sem == 2):

memID = listaMemorias[3].insertaReal()

elif(sem == 3):

memID = listaMemorias[3].insertaCaracter()

elif (sem == 5):

print("ERROR, los tipos de datos proveidos no son compatibles entre si.")

cuadruploList.normalCuad(op, oper1, oper2, memID)

#INSERTAR FUNCION DE CHEQUEO DE SEMANTICA, INSERTAR TEMPORALES ADECUADOS

stackOperando.append(memID)

indicetemporales = indicetemporales + 1

temporales.append(None)

#carga los operadores comparativos

def p\_ExpresAux(t):

'''

ExpresAux : OPERADOR\_COMPARATIVO

'''

global stackOperador

stackOperador.append(t[1])

#carga siguientes niveles de expresion

def p\_Exp(t):

'''

Exp : Exp ExpA

| Termino

'''

#carga el siguiente nivel de expresion

def p\_ExpA(t):

'''

ExpA : ExpAux Termino

'''

global stackOperador, stackOperando, cuadruploList, temporales, indicetemporales, checkSemantica,listaMemorias

top = stackOperador[len(stackOperador) - 1]

if (top == '+' or top == '-'):

temporales[indicetemporales] = "temporalExp"

op = stackOperador.pop()

oper2 = stackOperando.pop()

oper1 = stackOperando.pop()

memID = 0

sem = checkSemantica.Semantica(op,oper1, oper2)

if(sem == 0):

memID = listaMemorias[3].insertaBooleano()

elif(sem == 1):

memID = listaMemorias[3].insertaEntero()

elif(sem == 2):

memID = listaMemorias[3].insertaReal()

elif(sem == 3):

memID = listaMemorias[3].insertaCaracter()

elif (sem == 5):

print("ERROR, los tipos de datos proveidos no son compatibles entre si.")

cuadruploList.normalCuad(op, oper1, oper2, memID)

#INSERTAR FUNCION DE CHEQUEO DE SEMANTICA, INSERTAR TEMPORALES ADECUADOS

stackOperando.append(memID)

indicetemporales = indicetemporales + 1

temporales.append(None)

#carga los operadores de suma y resta

def p\_ExpAux(t):

'''

ExpAux : EXP\_OPERADOR

'''

global stackOperador

stackOperador.append(t[1])

#carga la siguientes partes de expresion

def p\_Termino(t):

'''

Termino : Termino TerminoA

| Factor

'''

#siguiente nivel de expresion compara

# el \* e / y general el cuadruplo correcto

def p\_TerminoA(t):

'''

TerminoA : TerminoAux Factor

'''

global stackOperador, stackOperando, cuadruploList, temporales, indicetemporales, checkSemantica,listaMemorias

top = stackOperador[len(stackOperador) - 1]

if (top == '\*' or top == '/'):

temporales[indicetemporales] = "temporalTermino"

op = stackOperador.pop()

oper2 = stackOperando.pop()

oper1 = stackOperando.pop()

sem = checkSemantica.Semantica(op,oper1, oper2)

memID = 0

if(sem == 0):

memID = listaMemorias[3].insertaBooleano()

elif(sem == 1):

memID = listaMemorias[3].insertaEntero()

elif(sem == 2):

memID = listaMemorias[3].insertaReal()

elif(sem == 3):

memID = listaMemorias[3].insertaCaracter()

elif (sem == 5):

print("ERROR, los tipos de datos proveidos no son compatibles entre si.")

#INSERTAR FUNCION DE CHEQUEO DE SEMANTICA, INSERTAR TEMPORALES ADECUADOS

cuadruploList.normalCuad(op, oper1, oper2, memID)

stackOperando.append(memID)

indicetemporales = indicetemporales + 1

temporales.append(None)

#inserta los terminos \* y / en el stack de operadores

def p\_TerminoAux(t):

'''

TerminoAux : TERM\_OPERADOR

'''

global stackOperador

stackOperador.append(t[1])

#Factor es el valor "atmopico de una operacion, el caul es el resultaod de algo entre parentesis"

# o de los elementos atomicos

def p\_Factor(t):

'''

Factor : ValorSalida

| ParentesisInit Expresion ParentesisFin

'''

# usar parentesis para meterlo como fondo falso

# usar parentesis para meterlo como fondo falso

#mete un fondo falso a la pila de operadores

def p\_ParentesisInit(t):

'''

ParentesisInit : PARENTESIS\_IZQ

'''

global stackOperador

stackOperador.append(t[1])

def p\_ParentesisFin(t):

'''

ParentesisFin : PARENTESIS\_DER

'''

global stackOperador

stackOperador.pop()

#atributos de llamda funcion para llamar adecuadamente a la funcion apropiadamente

def p\_LlamadaFuncion(t):

'''

LlamadaFuncion : INTER\_IZQ LlamadaFuncionA INTER\_DER FinalLlamada

'''

global paramCont

paramCont = 1

#diagrama de sintaxis para asignar multiples parametros a una llamada de funcion

def p\_LlamadaFuncionA(t):

'''

LlamadaFuncionA : Expresion CorreExpresion LlamadaFuncionB

| empty

'''

#genera cuadruplo de los parametros instanciados

def p\_CorreExpresion(t):

'''

CorreExpresion :

'''

global stackOperando, paramCont

op1 = stackOperando.pop()

texto = "param" + str(paramCont)

cuadruploList.normalCuad(texto, op1)

paramCont = paramCont + 1

#recursion para manejar los parametros que sean necesarios

def p\_LlamadaFuncionB(t):

'''

LlamadaFuncionB : COMMA LlamadaFuncionA

| empty

'''

#genera cuadruplo final

#cuadruplo gosub que te lleva a la funcion

def p\_FinalLlamada(t):

'''

FinalLlamada :

'''

global auxstackParam, cuadruploList,stackOperador

if (auxstackParam):

print ("ZOOL", auxstackParam)

auxstackParam.pop()

cuadruploList.normalCuad("Gosub", stackOperando.pop())

stackOperador.pop()

auxstackParam = []

#Diagrama para generar rapidamente las llamadas de parametro ,sean ,sencillas

#llamadas a funcion, o inclusive arreglos

def p\_DeclaraBase(t):

'''

DeclaraBase : Parametro DeclaraA

'''

global varLocal,tablaSimbolosActual,stackOperando,memoriaVirtual

varLocal = varLocal + 1

if(not(t[2] is None)):

limInf = t[2]['dimensionA']

limSup = t[2]['dimensionB']

tipo = t[1]

memID = 0

tipoID = 0

total = limInf\*limSup

inferior = total - 1

op = stackOperando.pop()

t[0] = op

existe = tablaSimbolosActual.buscar(op)

if(existe is None):

if(tablaSimbolosActual.padre is None):

if(not (tipo=='entero' or tipo =='booleano' or tipo =='caracter'or tipo =='real')):

existe = tablaGlobal.buscarHijos(t[1])

if( not (existe is None)):

#idValue = existe['memo']

memID = memoriaIDClases.insertaClase(existe.simbolos[t[1]]['memo'],total)

tablaSimbolosActual.insertar(op,t[1],memID)

##Sacar un id de clase, ver su tabla

##generar un id por elemento global

else:

print("TIPO DECLARADO NO EXISTENTE");

raise SyntaxError

else:

idValue = int(tablaSimbolosActual.id/10000)

if(tipo =='entero'):

memID = listaMemorias[idValue].insertaEntero(total)

elif(tipo =='booleano'):

memID = listaMemorias[idValue].insertaBooleano(total)

elif(tipo =='caracter'):

memID = listaMemorias[idValue].insertaCaracter(total)

elif(tipo =='real'):

memID = listaMemorias[idValue].insertaReal(total)

tablaSimbolosActual.insertar(op,tipo,memID,limInf,limSup)

memoriaVirtual.variablesDim[memID] = {'limInf':limInf,'limSup':limSup}

else:

existe = tablaSimbolosActual.padre.buscar(op)

if(existe is None):

if(not (tipo=='entero' or tipo =='booleano' or tipo =='caracter'or tipo =='real')):

existe = tablaGlobal.buscarHijos(t[1])

if( not (existe is None)):

#idValue = existe['memo']

memID = memoriaIDClases.insertaClase(existe.simbolos[t[1]]['memo'],total)

tablaSimbolosActual.insertar(op,t[1],memID,limInf,limSup)

memoriaVirtual.variablesDim[memID] = {'limInf':limInf,'limSup':limSup}

##Sacar un id de clase, ver su tabla

##generar un id por elemento global

else:

print("TIPO DECLARADO NO EXISTENTE");

raise SyntaxError

else:

idValue = int(tablaSimbolosActual.id/10000)

print("idValue a insertar",idValue)

if(tipo =='entero'):

memID = listaMemorias[idValue].insertaEntero(total)

elif(tipo =='booleano'):

memID = listaMemorias[idValue].insertaBooleano(total)

elif(tipo =='caracter'):

memID = listaMemorias[idValue].insertaCaracter(total)

elif(tipo =='real'):

memID = listaMemorias[idValue].insertaReal(total)

tablaSimbolosActual.insertar(op,tipo,memID,limInf,limSup)

memoriaVirtual.variablesDim[memID] = {'limInf':limInf,'limSup':limSup}

else:

print("VARIABLE PREVIAMENTE DECLARADA")

raise SyntaxError

else:

print("VARIABLE PREVIAMENTE DECLARADA")

raise SyntaxError

else:

tipo = t[1]

op = stackOperando.pop()

t[0] = op

existe = tablaSimbolosActual.buscar(op)

if(existe is None):

if(tablaSimbolosActual.padre is None):

if(not (tipo=='entero' or tipo =='booleano' or tipo =='caracter'or tipo =='real')):

existe = tablaGlobal.buscarHijos(t[1])

if( not (existe is None)):

#idValue = existe['memo']

memID = memoriaIDClases.insertaClase(existe.simbolos[t[1]]['memo'])

tablaSimbolosActual.insertar(op,t[1],memID)

##Sacar un id de clase, ver su tabla

##generar un id por elemento global

else:

print("TIPO DECLARADO NO EXISTENTE");

raise SyntaxError

else:

idValue = int(tablaSimbolosActual.id/10000)

if(tipo =='entero'):

memID = listaMemorias[idValue].insertaEntero()

elif(tipo =='booleano'):

memID = listaMemorias[idValue].insertaBooleano()

elif(tipo =='caracter'):

memID = listaMemorias[idValue].insertaCaracter()

elif(tipo =='real'):

memID = listaMemorias[idValue].insertaReal()

tablaSimbolosActual.insertar(op,tipo,memID)

else:

existe = tablaSimbolosActual.padre.buscar(op)

if(existe is None):

if(not (tipo=='entero' or tipo =='booleano' or tipo =='caracter'or tipo =='real')):

existe = tablaGlobal.buscarHijos(t[1])

if( not (existe is None)):

#idValue = existe['memo']

memID = memoriaIDClases.insertaClase(existe.simbolos[t[1]]['memo'])

tablaSimbolosActual.insertar(op,t[1],memID)

##Sacar un id de clase, ver su tabla

##generar un id por elemento global

else:

print("TIPO DECLARADO NO EXISTENTE");

raise SyntaxError

else:

idValue = int(tablaSimbolosActual.id/10000)

if(tipo =='entero'):

memID = listaMemorias[idValue].insertaEntero()

elif(tipo =='booleano'):

memID = listaMemorias[idValue].insertaBooleano()

elif(tipo =='caracter'):

memID = listaMemorias[idValue].insertaCaracter()

elif(tipo =='real'):

memID = listaMemorias[idValue].insertaReal()

tablaSimbolosActual.insertar(op,tipo,memID)

else:

print("VARIABLE PREVIAMENTE DECLARADA")

raise SyntaxError

else:

print("VARIABLE PREVIAMENTE DECLARADA")

raise SyntaxError

#iagrama de declaracion de variables a nivel global

#o dentor de un scope , pero no como atrbutos de una funcion

def p\_Declaracion(t):

'''

Declaracion : DeclaraBase SEMICOLON

'''

#diagram de sintaxis para un vector

def p\_DeclaraA(t):

'''

DeclaraA : CORCHETE\_IZQ CONST\_NUMERO\_ENT CORCHETE\_DER DeclaraB

| empty

'''

if(len(t) == 5):

t[0] = {'dimensionA':t[2],'dimensionB':t[4]['dimensionB']}

limInf = t[2]

elif(len(t) == 4):

t[0] = {'dimensionA':t[2],'dimensionB':1}

else:

t[0] = None

#diagrama de sintaxis para una matriz

def p\_DeclaraB(t):

'''

DeclaraB : CORCHETE\_IZQ CONST\_NUMERO\_ENT CORCHETE\_DER

| empty

'''

if(t[1] == '['):

t[0] = {"dimensionB":t[2]}

else:

t[0] = {"dimensionB":1}

#diagrama que devuele los valores atomicos de una expresion

def p\_ValorSalida(t):

'''

ValorSalida : NumeroEntero

| Caracter

| NumeroReal

| Booleano

| KEYWORD\_NULO

| LlamadaIDs

'''

#diagrama para la generaciond e llamadas id y llamadas de una funcion dentro para el uso de Valor Salida

def p\_LlamadaIDs(t):

'''

LlamadaIDs : LlamadaIDsAux LlamadaIDsA

'''

#Diagrama donde se anexa el identificador como una variable dentro del stack de operandos

def p\_LlamadaIDsAux(t):

'''

LlamadaIDsAux : IDENTIFICADOR

'''

global stackOperando, buscadorClase,pilaClase, procedimientoList, auxstackParam, tablaGlobal, cuadruploList, stackOperador

existe = None

existe = tablaSimbolosActual.buscar(t[1])

this = stackOperador[len(stackOperador)-1]

if (existe is None):

existe = tablaSimbolosActual.padre.buscar(t[1])

if (existe is None and (not (this == "."))):

print("El termino no ha sido declarado: ", t[1])

elif(not(existe is None)):

if(existe['tipo'] == 'real' or existe['tipo'] == 'booleano' or existe['tipo'] == 'caracter' or existe['tipo'] == 'entero'):

stackOperando.append(existe['memo'])

elif(existe['tipo'] =='funcion'):

auxstackParam.append(t[1])

auxstackParam.append(procedimientoList.buscar(t[1]))

cuadruploList.normalCuad("ERA", t[1])

#if (auxstackParam[1][0] is not None):

auxstackParam[1].reverse()

stackOperando.append(existe['memo'])

stackOperador.append("(")

stackOperando.append(t[1])

else:

buscadorClase = tablaGlobal.buscar(t[1])

pilaClase.append(t[1])

stackOperando.append(existe['memo'])

elif(not (len(stackOperador) == 0)):

if (this == '.'):

op1 = stackOperando.pop()

op = stackOperador.pop()

exis = tablaGlobal.buscar(op1)

if(exis is None):

print("Llamada no posible, clase no identificada")

raise SyntaxError

else:

cuadruploList.normalCuad(op, op1)

if(not(exis.buscar(t[1]) is None)):

exis = exis.buscar(t[1])

if(exis['tipo'] == 'real' or exis['tipo'] == 'entero' or exis['tipo'] == 'booleano' or exis['tipo'] == 'caracter'):

stackOperando.append(exis['memo'])

elif(exis['tipo'] == 'funcion'):

auxstackParam.append(exis['memo'])

auxstackParam.append(procedimientoList.buscar(t[1]))

cuadruploList.normalCuad("ERA", t[1])

#if (auxstackParam[1][0] is not None):

auxstackParam[1].reverse()

stackOperando.append(t[1])

stackOperador.append("(")

else:

#see t[1]

if(existe['tipo'] == 'real' or existe['tipo'] == 'booleano' or existe['tipo'] == 'caracter' or existe['tipo'] == 'entero'):

stackOperando.append(existe['memo'])

elif(existe['tipo'] == 'funcion'):

print("NO PUEDES GENERAR UNA FUNCION DENTRO DE OTRA")

raise SyntaxError

else:

buscadorClase = tablaGlobal.buscarHijos(existe['tipo'])

stackOperando.append(existe['memo'])

def p\_LlamadaIDsA(t):

'''

LlamadaIDsA : Terminal

| LlamadaFuncion

'''

#anexa constante entera a la tabla de simbolos,

#asigna valor de memoria ,y la mete al stack de operandos

def p\_NumeroEntero(t):

'''

NumeroEntero : CONST\_NUMERO\_ENT

'''

global tablaConstantes,stackOperando,listaMemorias

existe = None

existe = tablaConstantes.buscar(t[1])

if (existe is None):

memID = listaMemorias[2].insertaEntero()

tablaConstantes.insertar(t[1], "entero",memID)

stackOperando.append(memID)

else:

stackOperando.append(existe['memo'])

#anexa constante caracter a la tabla de simbolos,

#asigna valor de memoria ,y la mete al stack de operandos

def p\_Caracter(t):

'''

Caracter : CONST\_CARACTERES

'''

global tablaConstantes,stackOperando

existe = None

existe = tablaConstantes.buscar(t[1])

if (existe is None):

memID = listaMemorias[2].insertaCaracter()

tablaConstantes.insertar(t[1], "caracter",memID)

stackOperando.append(memID)

else:

stackOperando.append(existe['memo'])

#anexa constante real a la tabla de simbolos,

#asigna valor de memoria ,y la mete al stack de operandos

def p\_NumeroReal(t):

'''

NumeroReal : CONST\_NUMERO\_REAL

'''

global tablaConstantes,stackOperando

existe = None

existe = tablaConstantes.buscar(t[1])

if (existe is None):

memID = listaMemorias[2].insertaReal()

tablaConstantes.insertar(t[1], "real",memID)

stackOperando.append(memID)

else:

stackOperando.append(existe['memo'])

#anexa constante Booleana a la tabla de simbolos,

#asigna valor de memoria ,y la mete al stack de operandos

def p\_Booleano(t):

'''

Booleano : CONST\_BOOLEANO

'''

global tablaConstantes,stackOperando

existe = None

existe = tablaConstantes.buscar(t[1])

if (existe is None):

memID = listaMemorias[2].insertaBooleano()

tablaConstantes.insertar(t[1], "booleano",memID)

stackOperando.append(memID)

else:

stackOperando.append(existe['memo'])

#para manejar dentro de llamadas ids, arreglos , llamadas a funcion , llamadas a atributos de clase

def p\_Terminal(t):

'''

Terminal : AsignaClass

'''

#meter el punto como operador para accesar rapidamente las variables de clase y obtener las direcciones de memoria respectivas

def p\_PuntoAux(t):

'''

PuntoAux : PUNTO

'''

global stackOperador

stackOperador.append(t[1])

import ply.yacc as yacc

parser = yacc.yacc(start='Programa')

fileload = input('Nombre del archivo de entrada: ')

with open(fileload) as fileval:

result = parser.parse(fileval.read())

lexer.input(fileval.read())

# Tokenize

# while True:

# tok = lexer.token()

# if not tok:

# break # No more input

# print(tok)

print(result)

## maquina.py

from tablas import \*

from memoria import \*

class Maquina:

def \_\_init\_\_(self):

self.cuad = list()

self.proc = list()

self.simbolosMaquina = TablaSimbolos()

self.constantesMaquina = TablaConstantes()

self.memoriaTemporal = MemoriaReal(30001)

def setCuad(self, list=[]):

self.cuad = list

def setProc(self, list=[]):

self.proc = list

def setSimbolos(self, val):

self.simbolosMaquina = val

def setConstantes(self, val):

self.constantesMaquina = val

def setMemTemp(self, val):

self.memoriaTemporal = val

def calculos(self):

indiceCuad = 0

indiceRET = 0

listParam = []

last = len(self.cuad) - 1

listaMemorias = list()

memoriaVirtual = VirtualMemory("global")

simbol = self.simbolosMaquina

while(self.cuad[indiceCuad][0] != 'FIN'):

cuadru = self.cuad[indiceCuad]

if cuadru[0] == "+":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 + op2

print("suma!", result)

elif cuadru[0] == "-":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 - op2

print("resta!", result)

elif cuadru[0] == "\*":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 \* op2

print("multiplica!", result)

elif cuadru[0] == "/":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 / op2

print("divide!", result)

elif cuadru[0] == ">":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 > op2

if (result == True):

print("Verifica >!", "Verdadero")

else:

print("Verifica >!", "Falso")

elif cuadru[0] == "<":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (2499 < op1 < 5000):

op1 = 0

elif (op1 < 7500):

op1 = 0.0

elif(op1 < 20001):

op1 -= 10000

if (2499 < op1 < 5000):

op1 = 1

elif (op1 < 7500):

op1 = 0.1

elif(op1 < 30001):

op1 -= 20000

if (2499 < op1 < 5000):

op1 = 2

elif (op1 < 7500):

op1 = 0.2

else:

op1 -= 30000

if (2499 < op1 < 5000):

op1 = 3

elif (op1 < 7500):

op1 = 0.3

if(op2 < 10001):

if (2499 < op2 < 5000):

op2 = 0

elif (op2 < 7500):

op2 = 0.0

elif(op2 < 20001):

op2 -= 10000

if (2499 < op2 < 5000):

op2 = 1

elif (op2 < 7500):

op2 = 0.1

elif(op2 < 30001):

op2 -= 20000

if (2499 < op2 < 5000):

op2 = 2

elif (op2 < 7500):

op2 = 0.2

else:

op2 -= 30000

if (2499 < op2 < 5000):

op2 = 3

elif (op2 < 7500):

op2 = 0.3

result = op1 < op2

if (result == True):

print("Verifica <!", "Verdadero")

else:

print("Verifica <!", "Falso")

elif cuadru[0] == "&&":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (op1 < 2500):

op1 = True

elif(op1 < 20001):

if (op1 < 2500):

op1 = False

elif(op1 < 30001):

if (op1 < 2500):

op1 = True

else:

if (op1 < 2500):

op1 = False

if(op2 < 10001):

if (op2 < 2500):

op2 = True

elif(op2 < 20001):

if (op2 < 2500):

op2 = False

elif(op2 < 30001):

if (op2 < 2500):

op2 = True

else:

if (op2 < 2500):

op2 = False

result = op1 and op2

if (result == True):

print("Verifica &&!", "Verdadero")

else:

print("Verifica &&!", "Falso")

elif cuadru[0] == "||":

op1 = cuadru[1]

op2 = cuadru[2]

res = cuadru[3]

if(op1 < 10001):

if (op1 < 2500):

op1 = True

elif(op1 < 20001):

if (op1 < 2500):

op1 = False

elif(op1 < 30001):

if (op1 < 2500):

op1 = True

else:

if (op1 < 2500):

op1 = False

if(op2 < 10001):

if (op2 < 2500):

op2 = True

elif(op2 < 20001):

if (op2 < 2500):

op2 = False

elif(op2 < 30001):

if (op2 < 2500):

op2 = True

else:

if (op2 < 2500):

op2 = False

result = op1 or op2

if (result == True):

print("Verifica ||!", "Verdadero")

else:

print("Verifica ||!", "Falso")

elif cuadru[0] == "Goto":

res = cuadru[3]

indiceCuad = res - 1

print("INDICE Goto", indiceCuad+1)

elif cuadru[0] == "GotoF":

op1 = cuadru[1]

res = cuadru[3]

op1 = False

if op1 is False:

indiceCuad = res - 1

print("INDICE GotoF", indiceCuad+1)

elif cuadru[0] == "Gosub":

res = cuadru[1]

proIndex = 0

indiceRET = indiceCuad

while(self.proc[proIndex][0] != res):

proIndex = proIndex + 1

proced = self.proc[proIndex]

newIndex = proced[3]

indiceCuad = newIndex - 1

elif cuadru[0] == "RET":

indiceCuad = indiceRET

elif cuadru[0] == "ERA":

res = cuadru[1]

memID = 0

tipo = simbol.buscarTipo(res, "retorno")

idValue = int(self.simbolosMaquina.id/10000)

print("idValue a insertar",idValue)

if(tipo =='entero'):

memID = self.memoriaTemporal.insertaEntero()

elif(tipo =='booleano'):

memID = self.memoriaTemporal.insertaBooleano()

elif(tipo =='caracter'):

memID = self.memoriaTemporal.insertaCaracter()

elif(tipo =='real'):

memID = self.memoriaTemporal.insertaReal()

self.simbolosMaquina.insertar(res, tipo, memID)

memoriaVirtual.functions[memID] = {'funcion':res}

elif cuadru[0] == "ver":

op1 = cuadru[1]

op2 = cuadru[2]

op3 = cuadru[3]

if (op1 < op2) and (op1 > op3):

print("Se sale del rango de memoria")

indiceCuad = last

elif cuadru[0] == "salida":

op1 = cuadru[1]

print(op1)

indiceCuad += 1

print(memoriaVirtual.functions)

## memoria.py

#clase para Manejo de memoria virtual

#se dividen los diferentes tipos de objetos em

#directorios ya que es la manera mas rapida de accesarlos en python.

class VirtualMemory:

def \_\_init\_\_(self,name):

self.memory\_name = name

#variables locales de una funcion

self.functions = dict()

self.variables = dict()

self.variablesDim = dict()

self.clases = dict()

def insertaEntero(self,entero,capacidad):

if(entero < capacidad):

self.listaMemo[102].append(entero)

else:

print("Limite de memoria excedido")

def insertaReales(self,real,capacidad):

if(real < capacidad):

self.listaMemo[103].append(real)

else:

print("Limite de memoria excedido")

def insertaCaracteres(self,caracter,capacidad):

if(caracter < capacidad):

self.listaMemo[104].append(caracter)

else:

print("limite memoria excedido")

def imprimeMemoriaV(self):

print("Memoria de :" ,self.memory\_name)

print("booleanos en memoria:", self.listaMemo[101])

print("enteros en memoria:", self.listaMemo[102])

print("real en memoria:",self.listaMemo[103])

print("caracteres en memoria:",self.listaMemo[104])

#clase de memoria Real

#darle rango le da los valores de donde inicia y termina un tipo de memoria, ademas

# de ayudar a instanciar variables en memoria.

class MemoriaReal:

def \_\_init\_\_(self, rango = 0,clase = None):

self.booleanos = rango

self.enteros = rango + 2500

self.reales = rango + 5000

self.caracteres = rango + 7500

self.clase = clase

self.claseLista = dict()

self.cont\_bool = self.booleanos

self.cont\_ent = self.enteros

self.cont\_real = self.reales

self.cont\_car = self.caracteres

def insertaBooleano(self,tope = None):

if(tope is None):

if(self.clase is None):

if(self.cont\_bool < self.enteros):

self.cont\_bool = self.cont\_bool + 1

return self.cont\_bool

else:

print("memoria fuera de limites")

else :

self.cont\_bool = self.cont\_bool + 1

return self.cont\_bool

else:

min = self.cont\_bool + 1

top = self.cont\_bool + tope

if(top < self.enteros):

self.cont\_bool = self.cont\_bool + top

return self.cont\_bool

else:

print("memoria fuera de limites")

def insertaEntero(self,tope = None):

if(tope is None):

if(self.clase is None):

if(self.cont\_ent < self.reales):

self.cont\_ent = self.cont\_ent + 1

return self.cont\_ent

else:

print("memoria fuera de limites")

else:

print("memoria solo para clases")

else:

min = self.cont\_ent + 1

top = self.cont\_ent + tope

if(top < self.reales):

self.cont\_ent = self.cont\_ent + top

return self.cont\_ent

else:

print("memoria fuera de limites")

def insertaReal(self, tope = None):

if(tope is None):

if(self.clase is None):

if(self.cont\_real < self.caracteres):

self.cont\_real = self.cont\_real + 1

return self.cont\_real

else:

print("memoria fuera de limites")

else:

print("memoria Exclusiva de clases")

else:

min = self.cont\_real + 1

top = self.cont\_real + tope

if(top < self.caracteres):

self.cont\_real = self.cont\_real + top

return self.cont\_real

print("pasas por aqui no?")

else:

print("memoria fuera de limites")

def insertaCaracter(self,tope = None):

if(tope is None):

if(self.clase is None):

if(self.cont\_car < self.caracteres + 2500):

self.cont\_car = self.cont\_car + 1

return self.cont\_car

else:

print("memoria fuera de limites")

else:

print("memoria Exclusiva de clases")

else:

min = self.cont\_car + 1

top = self.cont\_car + tope

if(top < self.caracteres+2500):

self.cont\_car = self.cont\_car + top

return self.cont\_car

else:

print("memoria fuera de limites")

def insertaClase(self, claseID,tope = None):

claseAux = int(claseID / 10000)

claseAux = claseAux \* 10000

claseAux = claseAux + 500000

if (tope is None):

if (not(claseAux in self.claseLista)):

self.claseLista[claseAux] = claseAux

self.claseLista[claseAux] = claseAux + 1

return self.claseLista[claseAux]

else:

if(self.claseLista[claseAux] < (self.booleanos + claseAux + 10000)):

self.claseLista[claseAux] = self.claseLista[claseAux] + 1

return self.claseLista[claseAux]

else:

print("espacio excedido")

else:

if (not(claseAux in self.claseLista)):

self.claseLista[claseAux] = claseAux

self.claseLista[claseAux] = claseAux + tope

return self.claseLista[claseAux]

else:

if(self.claseLista[claseAux] < (self.booleanos + claseAux + 10000)):

self.claseLista[claseAux] = self.claseLista[claseAux] + tope

return self.claseLista[claseAux]

else:

print("espacio excedido")

def eliminaTemporales(self,topeBool, topeInt,topeReal, topeCar):

print("funcion de borrado")

## tablas.py

# cubo semantico es un diccionario de matrices que tiene de Id los tipos de operador que puede haber

# Ejemplo de como son cada una

# bool=0,int=1,float =2 ,string = 3,clase = 4,error = 5

# +, [

# bool [bool,int,float,string,clase],

# int [bool,int,float,string,clase],

# float [bool, int ,float,string, clase],

# string [bool, int ,float,string, clase],

# Clase [bool, int ,float,string, clase]

# ]

class claseCuboSemantico:

def \_\_init\_\_(self):

self.DataTypes = ['bool', 'int', 'real', 'caracter', 'clase', 'error']

self.Cubo = {'+': [[0, 1, 2, 5, 5], [1, 1, 2, 5, 5], [2, 2, 2, 5, 5], [5, 5, 5, 3, 5], [5, 5, 5, 5, 5]],

'-': [[0, 1, 2, 5, 5], [1, 1, 2, 5, 5], [2, 2, 2, 5, 5], [5, 5, 5, 4, 5], [5, 5, 5, 5, 5]],

'/': [[0, 1, 2, 5, 5], [1, 1, 2, 5, 5], [2, 2, 2, 5, 5], [5, 5, 5, 3, 5], [5, 5, 5, 5, 5]],

'\*': [[0, 1, 2, 5, 5], [1, 1, 2, 5, 5], [2, 2, 2, 5, 5], [5, 5, 5, 3, 5], [5, 5, 5, 5, 5]],

'=': [[0, 5, 5, 5, 5], [5, 1, 5, 5, 5], [5, 5, 2, 5, 5], [5, 5, 5, 4, 5], [5, 5, 5, 5, 5]],

'>': [[5, 5, 5, 5, 5], [5, 0, 0, 5, 5], [5, 0, 0, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5]],

'<': [[5, 5, 5, 5, 5], [5, 0, 0, 5, 5], [5, 0, 0, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5]],

'&&': [[0, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5]],

'||': [[0, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5], [5, 5, 5, 5, 5]],

'entrada': [[0, 5, 5, 5, 5], [5, 1, 5, 5, 5], [5, 5, 2, 5, 5], [5, 5, 5, 3, 5], [5, 5, 5, 5, 5]]

}

def Semantica(self, operador, operando1, operando2):

aux = int(operando1/10000)

aux2 = int(operando2/10000)

VerdaderoValor1 = operando1 - aux\*10000

VerdaderoValor2 = operando2 - aux2\*10000

IndexOP1 = 5

IndexOP2 = 5

if(VerdaderoValor1 >= 0 and VerdaderoValor1 <= 2500):

IndexOP1 = 0

elif(VerdaderoValor1 >= 2501 and VerdaderoValor1 <= 5000):

IndexOP1 = 1

elif(VerdaderoValor1 >= 5001 and VerdaderoValor1 <= 7500):

IndexOP1 = 2

elif(VerdaderoValor1 >= 7501 and VerdaderoValor1 <= 10000):

IndexOP1 = 3

if(VerdaderoValor2 >= 0 and VerdaderoValor2 <= 2500):

IndexOP2 = 0

elif(VerdaderoValor2 >= 2501 and VerdaderoValor2 <= 5000):

IndexOP2 = 1

elif(VerdaderoValor2 >= 5001 and VerdaderoValor2 <= 7500):

IndexOP2 = 2

elif(VerdaderoValor2 >= 7501 and VerdaderoValor2 <= 10000):

IndexOP2 = 3

if IndexOP1 < 5 and IndexOP2 < 5:

sem = self.Cubo[operador][IndexOP1][IndexOP2]

if sem == 5:

print("\nERROR TYPE MISMATCH. Los operandos:", operando1, "y", operando2,

"no son compatibles con el operador:", operador)

return 5

else:

return sem

else:

print("\nERROR. Tipos de datos:", operando1, ",", operando2, "y/o operador:", operador, "desconocidos.")

return None

# en esta clase se guardan las tabals constantes para facilitar su manejo en uso de maquina virutal

class TablaConstantes:

def \_\_init\_\_(self):

self.simbolos = dict()

def insertar(self, id, tipo, memID):

self.simbolos[id] = {'tipo':tipo,'memo':memID}

def buscar(self, id):

return self.simbolos.get(id)

def imprimir(self):

print ("Tabla de Constantes",self.simbolos)

#clase para generar las Tablas de simbolos , donde puedes insertar ,funciones

#clases, y variables. Ademas de buscar en estas clsaes esas mismas y usarlas

#como sea necesario

class TablaSimbolos:

def \_\_init\_\_(self,ident=0):

self.id = ident

self.simbolos = dict()

self.hijos = list()

self.padre = None

# agregar atributo name?

def insertar(self, id, tipo,memID,limInf = None,limSup = None):

if(limInf is None):

self.simbolos[id] = {'tipo':tipo, 'memo':memID}

else:

self.simbolos[id] = {'tipo':tipo,'memo':memID,'dimA':limInf,'dimB':limSup}

def convertirArreglo(self,id,idMax,limInf,limSup = None):

tipo = self.simbolos[id]['tipo']

memo = self.simbolos[id]['memo']

self.simbolos[id] = {'tipo':tipo,'memo':memo,'dimA':limInf,'dimB':limSup}

def insertarFuncion(self,id,tipo,memID):

self.simbolos[id] = {'tipo':'funcion', 'memo':memID,'retorno':tipo}

def insertarClase(self,id,memID,herencia = None):

if (herencia is None):

self.simbolos[id] = {'tipo':'clase','memo': memID,'id':id,'herencia':None}

else:

self.simbolos[id] = {'tipo':'clase','memo': memID,'id':id,'herencia':herencia}

def buscar(self, id):

return self.simbolos.get(id)

def buscarTipo(self, id, tipo):

return self.simbolos[id][tipo]

def agregarHijo(self, hijo):

self.hijos.append(hijo)

def agregarPadre(self, pad):

self.padre = pad

def devolverPadre(self):

if (self.padre is None):

print("no hay padre al cual ir");

else:

return self.padre

def buscarHijos(self, name):

for hijo in self.hijos:

existe = hijo.buscar(name)

if (existe is not None):

return hijo

# def \_\_str\_\_(self):

def imprimir(self):

i = 0

for hijo in self.hijos:

print("Hijo:",hijo.simbolos)

print ("tablaGlobal",self.simbolos)

#Clase para generar cuadruplos , lista de cuatro atributos , imprimible para ver si se guardaron

#los valores apropiadamente

class Cuadruplos:

def \_\_init\_\_(self):

self.cuadruplos = list()

def normalCuad(self, operador, operando1 = None, operando2 = None, destino=None):

self.cuadruplos.append((operador, operando1, operando2, destino))

print("operador:" ,operador , " op1:",operando1, " op2:", operando2 , " destino:",destino)

def updateCuad(self, index, operador=None, operando1=None, operando2=None, destino=None):

self.cuadruplos[index] = (operador, operando1, operando2, destino)

def AssignCuad(self,operador, operando1, destino):

self.cuadruplos.append((operador, operando1, None, destino))

def SaltaCuad(self, Goto, destino=None):

self.cuadruplos.append((Goto, None, None, destino))

return len(self.cuadruplos) - 1

def AgregarSalto(self, indice, expr, destino=None):

if destino is None:

destino = len(self.cuadruplos)

salto = (self.cuadruplos[indice][0], expr, None, destino)

self.cuadruplos[indice] = salto

def EspecialCuad(self, operador, operando1, operando2, destino):

print("cuadruplo a usar en funciones especiales")

def CuadSize(self):

return len(self.cuadruplos)

def Ultimo(self):

return self.cuadruplos[-1]

def imprimir(self):

indice = 0

for cuad in self.cuadruplos:

print('indice:', indice, 'operador: ', cuad[0], 'operando1: ', cuad[1], 'operando2: ', cuad[2], 'destino:',

cuad[3])

indice = indice + 1

def getCuadruplos(self):

return self.cuadruplos

class Procedimientos:

def \_\_init\_\_(self):

self.procedimientos = list()

self.listParam = dict()

def normalLista(self, id, parametros, variables, cuadruplo, scope):

self.procedimientos.append((id, parametros, variables, cuadruplo, scope))

print("ID Procedimiento:" ,id , " # Param:",parametros, " # Variables:", variables , "Destino:",cuadruplo, "Scope:",scope)

def updateLista(self, index, id, parametros, variables, destino, scope):

self.procedimientos[index] = (id, parametros, variables, destino, scope)

def meteParametros(self, id, lista = []):

self.listParam[id] = lista

def buscar(self, id):

return self.listParam.get(id)

def ListaSize(self):

return len(self.procedimientos)

def Ultimo(self):

return self.procedimientos[-1]

def imprimir(self):

indice = 0

for proc in self.procedimientos:

print('indice:', indice, 'ID Procedimiento: ', proc[0], '#Param: ', proc[1], '#Variables: ', proc[2], 'Destino:',

proc[3], 'Scope:', proc[4])

indice = indice + 1

def getProcedimientos(self):

return self.procedimientos

Manual de Usuario

Instalación de las herramientas requeridas

Tuby utiliza principalmente 3 herramientas para poder correr de manera adecuada. Las 3 son Python, PLY y Pycharm.

#### Python

Tuby está basado en el lenguaje de programación Python. Para poder instalar el lenguaje, puede utilizar el siguiente link en el que se encuentra la version mas nueva 3.5.2, la cual es en la que se desarrolló nuestro lenguaje. <https://www.python.org/downloads/>

#### PLY

PLY es una herramienta de python que utilizamos para poder desarrollar Tuby. Pueden utilizar el siguiente link de descarga para descargar la versión más reciente. <http://www.dabeaz.com/ply/>

Para hacerlo funcionar con el lenguaje, coloque la carpeta ply que extrajo en el directorio que se encuentran los archivo del lenguaje.

#### PyCharm

PyCharm es el IDE que utilizamos para correr y desarrollar el lenguaje. Por medio de Pycharm pudimos verificar que el lenguaje está corriendo adecuadamente. Utilize el siguiente link de descarga para poder descargar, utilizar PyCharm y poder utilizar Tuby ahora <https://www.jetbrains.com/pycharm/>.

Compilación

Para poder compilar su programa, abra scanner.py en PyCharm. De click en la barra de Menú llamada “Run”. Despues Run > scanner.py para que se compilen las intrucciones del compilador. Le pedirá el nombre de su archivo, teclee para poder completarlo.

Convenciones de Léxico

## Identificadores

#### Variables numericas

Se cuentan con 2 opciones de variable numéricas, las cuales son enteros o reales. Los enteros son números que no cuentan con punto decimal y los reales son números que pueden ser utilizados para operaciones decimales. Para definirlos hágalo de la siguiente manera:

**entero numero;**

**real numerodecimal;**

Para poder asignarles un valor, teclee el nombre que le dio, seguido de = y un valor numérico que sea aceptable en sus valores.

**numero = 2;**

**numerodecimal = 1.1;**

#### Variables de texto

Tuby cuenta con variables de texto llamadas caracteres, se definen igual que las numericas, pero utilizan la palabra clave caracter al inicio.

**Caracter nombre;**

Para poder asignarles un valor se hace de la siguiente manera.

**Nombre = Ruben;**

#### Variables Booleanas

Las variables Booleanas pueden se utilizadas para proporcionar valores de verdadero y falso a condiciones que queramos manejar. Se pueden utilizar para entrar o salir de condiciones o ciclos.

**booleano compruebaFalso;**

**booleano compruebaVerdadero;**

**booleano compruebaFalso = falso;**

**booleano compruebaVerdadero = verdadero;**

## Palabras reservadas

Entero, real, booleano, si', sino, mientras, clase, principal, caracter, entrada, salida, funcion, nulo, retorno, verdadero, falso

## Diseño de programas

Los programas están divididos en una sección para declarar tus clases, funciones o variables globales que quieran utilizar en el programa y la función principal que hará referencia a todas las funciones y variables que se le pide que referencia en sus instrucciones.

## Expresiones

Al realizar una expresión se debe tener compatibilidad entre las variables o constantes introducidas en la expresión.

**expresion1 = 2+3; O**

**expresion2 = expresion1 / 2; O**

**expresion3 = (4 > 5); O**

**expresion4 = “baile” + falso; X**

**expresion5 = verdadero - falso; X**

## Condiciones

Las condiciones nos ayudan a verificar cuando va a seguir o terminar un ciclo, dar pase para realizar ciertas acciones. Dentro de los paréntesis de las condiciones se tiene que introducir una expresión que decía las acciones a tomar dentro de ellas.

**booleano no = falso;**

**si (booleano){**

**contador = contador + 2;**

**}sino {**

**contador = 2;**

**}**

## Ciclos

Los ciclos que utilizamos tiene que ser iniciados con la palabra mientras, seguida de una expresión dentro de un paréntesis. Al momento de que la condición no se cumpla, la ejecucion del lenguaje saldra del bloque de mientras.

**mientras (hola < adios && adios > buenas noches){**

**baila[hola] = adios;**

**hola = hola + 1;**

**adios = adios + 1;**

**}**

## Funciones

Las funciones de declaran utilizando la palabra clave función, seguía por el tipo de valor que regresaran y después de su nombre, después tendrán que proveer los parámetros que maneja junto con sus tipos dentro de signos de interrogación ¿? y separados por comas y el bloque con las acciones a realizar.

**función entero perro ¿entero rojo?{**

**entero azul;**

**azul = rojo;**

**retorno azul + 4;**

**}**

## Clases

Las clases pueden ser utilizadas para declarar funciones propias que nadamas se utilizarán cuando se corren haciendo referencia a estas mismas clases. Sus bloques tienen que terminar en punto y coma para saber que está terminando la clase.

**clase Sayajin{**

**entero nivel\_de\_pelea;**

**booleano mono;**

**entero superSayajin;**

**funcion caracter dameSayajin¿?{**

**entero sol;**

**sol = superSayajin + 3;**

**retorno sol;**

**}**

**};**

Las clases también puede tener herencia de padres al proveer 2 puntos más el nombre de la clase padre justo después del nombre de la clase hijo, después de declararla.

**clase Goku:Sayajin{**

**entero gohan;**

**real vegeta;**

**booleano milk;**

**funcion booleano nombreMilk¿?{**

**entero azulado;**

**salida azulado + 5;**

**retorno milk;**

**}**

**};**

1. "Lenguas de México - Wikipedia, la enciclopedia libre." 2011. 23 Sep. 2016 <<https://es.wikipedia.org/wiki/Lenguas_de_M%C3%A9xico>> [↑](#footnote-ref-0)