Analizador sintáctico Ascendente

*'inicio : etiquetas'*

*'etiquetas : etiquetas etiqueta'*

*'etiquetas : etiqueta'*

*'etiqueta : IDENTIFICADOR ideti instrucciones'*

*'etiqueta : MAIN ideti instrucciones'*

*'ideti : DOBLEPUNTO'*

*'instrucciones : instrucciones instruccion '*

*'instrucciones : '*

*'instruccion : EXIT PUNTOCOMA'*

*'instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA'*

*'instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA'  
'instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA'*

*'instruccion : GOTO IDENTIFICADOR PUNTOCOMA'*

*'arra : CORA expresion CORC arra '  
'arra : '*

*'''expresion : valor MAS valor  
 | valor MENOS valor  
 | valor POR valor  
 | valor DIVIDIDO valor  
 | valor MOD valor  
 | valor ANDB valor  
 | valor ORB valor  
 | valor XORB valor  
 | valor SHIFTD valor  
 | valor SHIFTI valor  
  
  
  
  
 | valor AND valor  
 | valor OR valor  
 | valor XOR valor  
  
  
 | valor DIFERENTE valor  
 | valor IGUALDOBLE valor  
 | valor MAYOR valor  
 | valor MAYORIGUAL valor  
 | valor MENORIGUAL valor  
 | valor MENOR valor'''*

*'expresion : valor'*

*'''expresion : MENOS valor  
 | ABS PARA valor PARC  
 | NOTB valor  
 | NOT valor  
 | MAS valor  
 | ANDB DOLAR IDENTIFICADOR arra  
 | READ PARA PARC  
 | ARRAY PARA PARC '''  
'''expresion : PARA INT PARC valor  
 | PARA FLOAT PARC valor  
 | PARA CHAR PARC valor'''*

*'expresion : PARA expresion PARC'*

*'valor : ENTERO'*

*'valor : DECIMAL'*

*'valor : CADENA'*

*'valor : DOLAR IDENTIFICADOR arra'*

Analizador sintáctico Descendente

*'inicio : etiquetas'*

*'etiquetas : etiqueta etiquetasp'*

*'etiquetasp : etiqueta etiquetasp'*

*'etiquetasp : '*

*'etiqueta : IDENTIFICADOR ideti instrucciones'*

*'etiqueta : MAIN ideti instrucciones'*

*'ideti : DOBLEPUNTO'*

*'instrucciones : instruccionesp '*

*'instruccionesp : instruccion instruccionesp'*

*'instruccionesp : '*

*'instruccion : EXIT PUNTOCOMA'*

*'instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA'*

*'instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA'  
'instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA'*

*'instruccion : GOTO IDENTIFICADOR PUNTOCOMA'*

*'arra : CORA expresion CORC arra '  
'arra : '*

*'''expresion : valor MAS valor  
 | valor MENOS valor  
 | valor POR valor  
 | valor DIVIDIDO valor  
 | valor MOD valor  
 | valor ANDB valor  
 | valor ORB valor  
 | valor XORB valor  
 | valor SHIFTD valor  
 | valor SHIFTI valor  
  
  
  
  
 | valor AND valor  
 | valor OR valor  
 | valor XOR valor  
  
  
 | valor DIFERENTE valor  
 | valor IGUALDOBLE valor  
 | valor MAYOR valor  
 | valor MAYORIGUAL valor  
 | valor MENORIGUAL valor  
 | valor MENOR valor'''*

*'expresion : valor'*

*'''expresion : MENOS valor  
 | ABS PARA valor PARC  
 | NOTB valor  
 | NOT valor  
 | MAS valor  
 | ANDB DOLAR IDENTIFICADOR arra  
 | READ PARA PARC  
 | ARRAY PARA PARC '''  
'''expresion : PARA INT PARC valor  
 | PARA FLOAT PARC valor  
 | PARA CHAR PARC valor'''*

*'expresion : PARA expresion PARC'*

*'valor : ENTERO'*

*'valor : DECIMAL'*

*'valor : CADENA'*

*'valor : DOLAR IDENTIFICADOR arra'*

Analizador Léxico

# Tokens  
t\_PARA = r'\('  
t\_PARC = r'\)'  
t\_CORA = r'\['  
t\_CORC = r'\]'  
t\_MAS = r'\+'  
t\_MENOS = r'-'  
t\_POR = r'\\*'  
t\_DIVIDIDO = r'/'  
t\_PUNTOCOMA = r';'  
t\_DOBLEPUNTO = r':'  
t\_DOLAR = r'\$'  
t\_IGUAL = r'\='  
t\_MOD = r'\%'  
t\_NOT = r'\!'  
t\_AND = r'\&\&'  
t\_OR = r'\|\|'  
  
t\_NOTB = r'\~'  
t\_ANDB = r'\&'  
t\_ORB = r'\|'  
t\_XORB = r'\^'  
t\_SHIFTI = r'<<'  
t\_SHIFTD = r'>>'  
  
t\_IGUALDOBLE = r'\=\='  
t\_DIFERENTE = r'\!\='  
t\_MAYOR = r'\>'  
t\_MENOR = r'\<'  
t\_MENORIGUAL = r'\<\='  
t\_MAYORIGUAL = r'\>\='  
  
  
# Expresiones Regulares  
def t\_IDENTIFICADOR(t):  
 *r'[a-zA-Z\_][a-zA-Z\_0-9]\*'* t.type = reserved.get(t.value, 'IDENTIFICADOR') # Check for reserved words  
 return t  
  
  
def t\_DECIMAL(t):  
 *r'\d+\.\d+'* try:  
 t.value = float(t.value)  
 except ValueError:  
 print("Floaat value too large %d", t.value)  
 t.value = 0  
 return t  
  
  
def t\_ENTERO(t):  
 *r'\d+'* try:  
 t.value = int(t.value)  
 except ValueError:  
 print("Integer value too large %d", t.value)  
 t.value = 0  
 return t  
  
  
def t\_CADENA(t):  
 *r'("|\')([^"\']\*)("|\')'* try:  
 cadena: str = t.value  
 t.value = cadena.replace("\"", "")  
 t.value = t.value.replace("\'", "")  
 except ValueError:  
 print("Error Cadena %d", t.value)  
 t.value = ""  
 return t  
  
  
# Caracteres ignorados  
t\_ignore = " \t"  
  
  
def t\_newline(t):  
 *r'\n+'* t.lexer.lineno += t.value.count("\n")  
  
  
def t\_comentario(t):  
 *r'\#(.\*)(\n)?'* t.lexer.lineno += t.value.count("\n")

Analizador sintáctico Ascendente

# Asociación de operadores y precedencia  
from Contenido.LstInstruccion import ABCInstruccion  
from Contenido.LstInstruccion.Instruccion.Etiqueta import Etiqueta  
from Contenido.LstInstruccion.Instruccion.Exit import Exit  
from Contenido.LstInstruccion.Instruccion.Unset import Unset  
from Contenido.LstInstruccion.Registro.Asignar import Asignar  
from Contenido.LstInstruccion.Instruccion.goto import Goto  
from Contenido.LstInstruccion.Instruccion.If import If  
from Contenido.LstInstruccion.Registro.VariableValor import VariableValor  
from Contenido.LstInstruccion.Instruccion.Referencia import Referencia  
from Contenido.LstInstruccion.ABCInstruccion import Ts  
from .Lexico import \*  
import ply.lex as lex  
import ply.yacc as yacc  
  
precedence = (  
 ('left', 'MAS', 'MENOS'),  
 ('left', 'POR', 'DIVIDIDO'),  
  
)  
  
  
# Definición de la gramática  
def p\_inicio(t):  
 *'inicio : etiquetas'* t[0] = t[1]  
 global Ts  
 Ts.rp\_nuevo\_nodo('inicio : etiquetas',"t[0] = t[1]")  
  
def p\_etiquetas\_lista(t):  
 *'etiquetas : etiquetas etiqueta'* t[0] = t[1]  
 t[0].agregar(t[2])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('etiquetas : etiquetas etiqueta', "t[0] = t[1] \n t[0].agregar(t[2]) ")  
  
def p\_etiquetas\_lista\_inicio(t):  
 *'etiquetas : etiqueta'* t[0] = ABCInstruccion.ListaEtiqueta([t[1]])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('etiquetas : etiqueta', "t[0] = ABCInstruccion.ListaEtiqueta([t[1]])")  
  
  
def p\_etiqueta\_contenido(t):  
 *'etiqueta : IDENTIFICADOR ideti instrucciones'* t[0] = Etiqueta(t[3], t[1])  
 global Ts  
 Ts.consolidar\_etiqueta()  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('etiqueta : IDENTIFICADOR ideti instrucciones', "t[0] = Etiqueta(t[3], t[1])")  
  
def p\_etiqueta\_id\_error(t):  
 *'ideti : DOBLEPUNTO'* global Ts  
 Ts.nueva\_etiqueta(t[-1])  
  
 Ts.rp\_nuevo\_nodo('ideti : DOBLEPUNTO', "Ts.nueva\_etiqueta(t[-1]))")  
  
def p\_etiqueta\_principal(t):  
 *'etiqueta : MAIN ideti instrucciones'* t[0] = Etiqueta(t[3], t[1])  
 global Ts  
 Ts.consolidar\_etiqueta()  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('etiqueta : MAIN ideti instrucciones', "t[0] = Etiqueta(t[3], t[1])")  
  
def p\_instrucciones\_lista(t):  
 *'instrucciones : instrucciones instruccion '* t[0] = t[1]  
 t[0].agregar(t[2])  
 global Ts  
 Ts.nueva\_instruaccion(t[2])  
 Ts.rp\_nuevo\_nodo('instrucciones : instrucciones instruccion ', "t[0] = Etiqueta(t[3], t[1])")  
  
def p\_instrucciones\_lista\_inicio(t):  
 *'instrucciones : '* t[0] = ABCInstruccion.ListaInstruccion([])  
 global Ts  
 Ts.rp\_nuevo\_nodo('instrucciones : ', "t[0] = ABCInstruccion.ListaInstruccion([])")  
  
  
def p\_instrucciones\_exit(t):  
 *'instruccion : EXIT PUNTOCOMA'* t[0] = Exit()  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : EXIT PUNTOCOMA', "t[0] = Exit()")  
  
def p\_instrucciones\_imprimir(t):  
 *'instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA'* t[0] = ABCInstruccion.Imprimir(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA', "t[0] = ABCInstruccion.Imprimir(t[3])")  
  
def p\_instrucciones\_unset(t):  
 *'instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA'* t[0] = Unset(t[4])  
 t[0].indices(t[5])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA', "t[0] = Unset(t[4])")  
  
def p\_instrucciones\_asignar(t):  
 *'instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA'* t[0] = Asignar(t[2], t[5])  
 t[0].indices(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA', "t[0] = Asignar(t[2], t[5])")  
  
def p\_arreglo\_indice(t):  
 *'arra : CORA expresion CORC arra '* t[0] = t[4]  
 t[0].append(t[2])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('arra : CORA expresion CORC arra ', "t[0] = t[4] t[0].append(t[2])")  
  
def p\_arreglo\_indice\_epsilon(t):  
 *'arra : '* t[0] = []  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('arra : ', "t[0] = []")  
  
def p\_instrucciones\_goto(t):  
 *'instruccion : GOTO IDENTIFICADOR PUNTOCOMA'* t[0] = Goto(t[2])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : GOTO IDENTIFICADOR PUNTOCOMA', "t[0] = Goto(t[2])")  
  
def p\_instrucciones\_if(t):  
 *'instruccion : IF expresion instruccion '* t[0] = If(t[2], t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : IF expresion instruccion ', "t[0] = If(t[2], t[3])")  
  
# HASTA AQUI HAY GRAFICA  
def p\_expresion\_binaria(t):  
 *'''expresion : valor MAS valor  
 | valor MENOS valor  
 | valor POR valor  
 | valor DIVIDIDO valor  
 | valor MOD valor  
 | valor ANDB valor  
 | valor ORB valor  
 | valor XORB valor  
 | valor SHIFTD valor  
 | valor SHIFTI valor  
  
  
  
  
 | valor AND valor  
 | valor OR valor  
 | valor XOR valor  
  
  
 | valor DIFERENTE valor  
 | valor IGUALDOBLE valor  
 | valor MAYOR valor  
 | valor MAYORIGUAL valor  
 | valor MENORIGUAL valor  
 | valor MENOR valor'''* t[0] = ABCInstruccion.ExpresionDoble(t[1], t[2], t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[2]))  
 Ts.rp\_nuevo\_nodo('expresion : valor '+str(t[2])+ ' valor', "t[0] = ABCInstruccion.ExpresionDoble(t[1], t[2], t[3])")  
  
def p\_expresion\_sola(t):  
 *'expresion : valor'* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('expresion : valor',"t[0] = t[1]")  
  
def p\_expresion\_unaria(t):  
 *'''expresion : MENOS valor  
 | ABS PARA valor PARC  
 | NOTB valor  
 | NOT valor  
 | MAS valor  
 | ANDB DOLAR IDENTIFICADOR arra  
 | READ PARA PARC  
 | ARRAY PARA PARC '''* if t[2] == "(":  
 t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[3], t[1])  
 elif t[1] == "&":  
 t[0] = Referencia(t[3], t[4])  
  
 else:  
 t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[2], t[1])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('expresion : '+str(t[1])+'valor', "t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[3], t[1])")  
  
  
def p\_expresion\_agrupacion(t):  
 *'''expresion : PARA INT PARC valor  
 | PARA FLOAT PARC valor  
 | PARA CHAR PARC valor'''* t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[4], t[2])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('expresion : PARA' + str(t[2]) + 'PARC valor', "t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[4], t[2])")  
  
def p\_expresion\_parentesis(t):  
 *'expresion : PARA expresion PARC'* t[0] = t[2]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('expresion : PARA expresion PARC',"t[0] = t[2]")  
  
  
def p\_expresion\_entero(t):  
 *'valor : ENTERO'* t[0] = ABCInstruccion.Valor(t[1], 0)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : ENTERO', "t[0] = ABCInstruccion.Valor(t[1], 0)")  
  
def p\_expresion\_decimal(t):  
 *'valor : DECIMAL'* t[0] = ABCInstruccion.Valor(t[1], 1)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : DECIMAL', "t[0] = ABCInstruccion.Valor(t[1], 1)")  
  
def p\_expresion\_cadena(t):  
 *'valor : CADENA'* t[0] = ABCInstruccion.Valor(t[1], 2)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : CADENA', "t[0] = ABCInstruccion.Valor(t[1], 2)")  
  
  
def p\_expresion\_valor\_unico\_variable(t):  
 *'valor : DOLAR IDENTIFICADOR arra'* t[0] = VariableValor(t[2])  
 t[0].indices(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : DOLAR IDENTIFICADOR arra', "t[0] = VariableValor(t[2])")  
  
def find\_column(input, token):  
 if token is None:  
 return (0,0)  
 line\_start = input.rfind('\n', 0, token.lexpos) + 1  
 return ((token.lexpos - line\_start) + 1, token.lineno)  
  
  
def p\_error(t):  
 global Ts  
 Ts.exit\_exec = 0  
 tup=find\_column(Ts.texto\_analisis, t)  
 Ts.cargar\_error("El token con lexema \""+str(t.value)+"\" de tipo \""+str(t.type)+" \" ocasiono un error sintactico",20,tup)  
 print("Error sintáctico en '%s'" % t)  
  
  
def analizar\_ascendente(input: str):  
 # Construyendo el analizador léxico  
 lexer = lex.lex()  
 parser = yacc.yacc()  
 return parser.parse(input)

Analizador sintáctico Descendente

# Asociación de operadores y precedencia  
from Contenido.LstInstruccion import ABCInstruccion  
from Contenido.LstInstruccion.Instruccion.Etiqueta import Etiqueta  
from Contenido.LstInstruccion.Instruccion.Exit import Exit  
from Contenido.LstInstruccion.Instruccion.Unset import Unset  
from Contenido.LstInstruccion.Registro.Asignar import Asignar  
from Contenido.LstInstruccion.Instruccion.goto import Goto  
from Contenido.LstInstruccion.Instruccion.If import If  
from Contenido.LstInstruccion.Registro.VariableValor import VariableValor  
from Contenido.LstInstruccion.Instruccion.Referencia import Referencia  
from Contenido.LstInstruccion.ABCInstruccion import Ts  
from .Lexico import \*  
import ply.lex as lex  
import ply.yacc as yacc  
  
precedence = (  
 ('left', 'MAS', 'MENOS'),  
 ('left', 'POR', 'DIVIDIDO'),  
  
)  
  
  
# Definición de la gramática  
def p\_inicio(t):  
 *'inicio : etiquetas'* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('inicio : etiquetas', "t[0] = t[1]")  
  
  
def p\_etiquetas\_lista(t):  
 *'etiquetas : etiqueta etiquetasp'* t[0] = t[2]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('etiquetas : etiquetas etiqueta', "t[0] = t[2] ")  
  
def p\_etiqeutas\_sin\_recursividad(t):  
 *'etiquetasp : etiqueta etiquetasp'* t[0]=t[2]  
 t[0].lst.insert(0, t[-1])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('etiquetasp : etiqueta etiquetasp', "t[0] = t[2] \n t[0].lst.insert(0, t[-1])")  
  
def p\_etiqeutas\_sin\_recursividad\_epsilon(t):  
 *'etiquetasp : '* t[0] = ABCInstruccion.ListaEtiqueta([t[-1]])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('etiquetasp : ', "t[0] = ABCInstruccion.ListaEtiqueta([t[-1]])")  
  
def p\_etiqueta\_contenido(t):  
 *'etiqueta : IDENTIFICADOR ideti instrucciones'* t[0] = Etiqueta(t[3], t[1])  
 global Ts  
 Ts.consolidar\_etiqueta()  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('etiqueta : IDENTIFICADOR ideti instrucciones', "t[0] = Etiqueta(t[3], t[1])")  
  
  
def p\_etiqueta\_id\_error(t):  
 *'ideti : DOBLEPUNTO'* global Ts  
 Ts.nueva\_etiqueta(t[-1])  
  
 Ts.rp\_nuevo\_nodo('ideti : DOBLEPUNTO', "Ts.nueva\_etiqueta(t[-1])")  
  
def p\_etiqueta\_principal(t):  
 *'etiqueta : MAIN ideti instrucciones'* t[0] = Etiqueta(t[3], t[1])  
 global Ts  
 Ts.consolidar\_etiqueta()  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('etiqueta : MAIN ideti instrucciones', "t[0] = Etiqueta(t[3], t[1])")  
  
def p\_instrucciones\_lista(t):  
 *'instrucciones : instruccionesp '* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('instrucciones : instruccionesp ', "t[0] = t[1]")  
  
def p\_instrucciones\_lista\_sin\_recursividad(t):  
 *'instruccionesp : instruccion instruccionesp'* t[0] = t[2]  
 t[0].lst.insert(0, t[1])  
 global Ts  
 Ts.nueva\_instruaccion(t[1])  
  
 Ts.rp\_nuevo\_nodo('instruccionesp : instruccion instruccionesp', "t[0] = t[2] \n t[0].lst.insert(0, t[1])")  
  
def p\_instrucciones\_lista\_sin\_recursividad\_epsilon(t):  
 *'instruccionesp : '* t[0] = ABCInstruccion.ListaInstruccion([])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('instruccionesp : ', "t[0] = ABCInstruccion.ListaInstruccion([])")  
  
  
def p\_instrucciones\_exit(t):  
 *'instruccion : EXIT PUNTOCOMA'* t[0] = Exit()  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('instruccion : EXIT PUNTOCOMA', "t[0] = Exit()")  
  
def p\_instrucciones\_imprimir(t):  
 *'instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA'* t[0] = ABCInstruccion.Imprimir(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('instruccion : IMPRIMIR PARA expresion PARC PUNTOCOMA', "t[0] = ABCInstruccion.Imprimir(t[3])")  
  
  
def p\_instrucciones\_unset(t):  
 *'instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA'* t[0] = Unset(t[4])  
 t[0].indices(t[5])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('instruccion : UNSET PARA DOLAR IDENTIFICADOR arra PARC PUNTOCOMA', "t[0] = Unset(t[4])")  
  
  
def p\_instrucciones\_asignar(t):  
 *'instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA'* t[0] = Asignar(t[2], t[5])  
 t[0].indices(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('instruccion : DOLAR IDENTIFICADOR arra IGUAL expresion PUNTOCOMA', "t[0] = Asignar(t[2], t[5])")  
  
  
def p\_arreglo\_indice(t):  
 *'arra : CORA expresion CORC arra '* t[0] = t[4]  
 t[0].append(t[2])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('arra : CORA expresion CORC arra ', "t[0] = t[4]")  
  
  
def p\_arreglo\_indice\_epsilon(t):  
 *'arra : '* t[0] = []  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('arra : ', "t[0] = []")  
  
  
def p\_instrucciones\_goto(t):  
 *'instruccion : GOTO IDENTIFICADOR PUNTOCOMA'* t[0] = Goto(t[2])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('instruccion : GOTO IDENTIFICADOR PUNTOCOMA', "t[0] = Goto(t[2])")  
  
def p\_instrucciones\_if(t):  
 *'instruccion : IF expresion instruccion '* t[0] = If(t[2], t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
  
 Ts.rp\_nuevo\_nodo('instruccion : IF expresion instruccion ', "t[0] = If(t[2], t[3])")  
  
  
# HASTA AQUI HAY GRAFICA  
def p\_expresion\_binaria(t):  
 *'''expresion : valor MAS valor  
 | valor MENOS valor  
 | valor POR valor  
 | valor DIVIDIDO valor  
 | valor MOD valor  
 | valor ANDB valor  
 | valor ORB valor  
 | valor XORB valor  
 | valor SHIFTD valor  
 | valor SHIFTI valor  
  
  
  
  
 | valor AND valor  
 | valor OR valor  
 | valor XOR valor  
  
  
 | valor DIFERENTE valor  
 | valor IGUALDOBLE valor  
 | valor MAYOR valor  
 | valor MAYORIGUAL valor  
 | valor MENORIGUAL valor  
 | valor MENOR valor'''* t[0] = ABCInstruccion.ExpresionDoble(t[1], t[2], t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[2]))  
 Ts.rp\_nuevo\_nodo('expresion : valor ' + str(t[2]) + ' valor',  
 "t[0] = ABCInstruccion.ExpresionDoble(t[1], t[2], t[3])")  
  
  
def p\_expresion\_sola(t):  
 *'expresion : valor'* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('expresion : valor', "t[0] = t[1]")  
  
  
def p\_expresion\_unaria(t):  
 *'''expresion : MENOS valor  
 | ABS PARA valor PARC  
 | NOTB valor  
 | NOT valor  
 | MAS valor  
 | ANDB DOLAR IDENTIFICADOR arra  
 | READ PARA PARC  
 | ARRAY PARA PARC '''* if t[2] == "(":  
 t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[3], t[1])  
 elif t[1] == "&":  
 t[0] = Referencia(t[3], t[4])  
  
 else:  
 t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[2], t[1])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('expresion : ' + str(t[1]) + 'valor', "t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[3], t[1])")  
  
  
def p\_expresion\_agrupacion(t):  
 *'''expresion : PARA INT PARC valor  
 | PARA FLOAT PARC valor  
 | PARA CHAR PARC valor'''* t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[4], t[2])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('expresion : PARA' + str(t[2]) + 'PARC valor',  
 "t[0] = ABCInstruccion.ExpresionSimpleOperacion(t[4], t[2])")  
  
  
def p\_expresion\_parentesis(t):  
 *'expresion : PARA expresion PARC'* t[0] = t[2]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('expresion : PARA expresion PARC', "t[0] = t[2]")  
  
  
def p\_expresion\_entero(t):  
 *'valor : ENTERO'* t[0] = ABCInstruccion.Valor(t[1], 0)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : ENTERO', "t[0] = ABCInstruccion.Valor(t[1], 0)")  
  
  
def p\_expresion\_decimal(t):  
 *'valor : DECIMAL'* t[0] = ABCInstruccion.Valor(t[1], 1)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : DECIMAL', "t[0] = ABCInstruccion.Valor(t[1], 1)")  
  
  
def p\_expresion\_cadena(t):  
 *'valor : CADENA'* t[0] = ABCInstruccion.Valor(t[1], 2)  
 t[0] = ABCInstruccion.ExpresionSimple(t[0])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : CADENA', "t[0] = ABCInstruccion.Valor(t[1], 2)")  
  
  
def p\_expresion\_valor\_unico\_variable(t):  
 *'valor : DOLAR IDENTIFICADOR arra'* t[0] = VariableValor(t[2])  
 t[0].indices(t[3])  
  
 global Ts  
 t[0].n\_t(find\_column(Ts.texto\_analisis, t.slice[1]))  
 Ts.rp\_nuevo\_nodo('valor : DOLAR IDENTIFICADOR arra', "t[0] = VariableValor(t[2])")  
  
  
def find\_column(input, token):  
 line\_start = input.rfind('\n', 0, token.lexpos) + 1  
 return ((token.lexpos - line\_start) + 1, token.lineno)  
  
  
def p\_error(t):  
 global Ts  
 Ts.exit\_exec = 0  
 tup = find\_column(Ts.texto\_analisis, t)  
 Ts.cargar\_error(  
 "El token con lexema \"" + str(t.value) + "\" de tipo \"" + str(t.type) + " \" ocasiono un error sintactico",  
 20, tup)  
 print("Error sintáctico en '%s'" % t)  
  
  
def analizar\_descendente(input: str):  
 # Construyendo el analizador léxico  
 lexer = lex.lex()  
 parser = yacc.yacc()  
 return parser.parse(input)