Manual tecnico

Manual del proyecto 1 compiladores 2

201700886

**Herramientas Utilizadas**

**Python**

Lenguaje de programación utilizado para el desarrollo del programa, las ventajas que ofrece Python es su flexibilidad al ser multiparadigma y su débil tipado que ayuda a una programación más rápida.

**Pydot**

Librería que se encarga de generar código apto para Graphviz, su principal ventaja es que permite llamar al motor Graphviz, realizar la ejecución y devuelve en cadena de bytes la imagen generada, esa imagen generada se puede mostrar con un visor de imágenes dentro de la aplicación.

**Graphviz**

Es un programa que se encarga de generar grafos con base a un archivo de entrada.

**Pyqt5**

Es una librería que permite la creación de interfaces gráficas, además se pueden desarrollar desde el sistema drag and Drop.

**Explicación De Las Gramáticas**

# 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]")

Es la producción principal que se retorna como raíz cuando termina el análisis, la regla semántica solo se encarga de pasar la raíz que fue generada por la lista de etiquetas, el método rp nuevo nodo se encarga de agregar esa acción al reporte gramatical  
  
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]) ")

Reconoce una serie de etiquetas, las reg t[0]=t[1] sintetiza la lista de etiquetas que se llevan hasta el momento Ts.rp\_nuevo\_nodo crea la represntacion gramatical para la producción y el agregar pone el nuevo elemento en la lista  
  
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]])")

Reconoce el primer elemento de las listas de etiquetas y sintetiza una lista recien creada además crea la representación gramatical correspondiente  
  
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])")

Reconoce una etiqueta que tenga cualquier identificador como nombre, además posee una lista de instrucciones, se sintetiza una nueva etiqueta con una lista de instrucciones y el nombre que deberi tener, el método consolidar etiqueta permite guardar las instrucciones de las etiquetas sii se llega a presentar un error sintactico y por ultimo crea el reporte gramatical

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]))")

esta producción se encarga de inicializar una etiqueta por si se llegara a dar un error sintactico, y asi poder graficar la estructura del arbol  
  
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])")

Reconoce una etiqueta que tenga MAIN como nombre, además posee una lista de instrucciones, se sintetiza una nueva etiqueta con una lista de instrucciones y el nombre que deberi tener, el método consolidar etiqueta permite guardar las instrucciones de las etiquetas sii se llega a presentar un error sintactico y por ultimo crea el reporte gramatical

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])")

Reconoce una lista d einstrucciones agrega las nuevas instrucciones a la lista y genera reporte gramatical, agrega las instrucciones a la lista de instrucciones temporales  
  
def p\_instrucciones\_lista\_inicio(t):  
 *'instrucciones : '* t[0] = ABCInstruccion.ListaInstruccion([])  
 global Ts  
 Ts.rp\_nuevo\_nodo('instrucciones : ', "t[0] = ABCInstruccion.ListaInstruccion([])")

Reconoce una lista d einstrucciones agrega las nuevas instrucciones a la lista y genera reporte gramatical, agrega las instrucciones a la lista de instrucciones temporales,inicializa la lista de instrucciones

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()")

reconoce la instrucción exit y crea la hoja correspondiente y por ultimo agrega al reporte gramatical  
  
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])")

reconoce la instrucción imprimiry crea la hoja correspondiente y por ultimo agrega al reporte gramatical

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])")

reconoce la instrucción unset y crea la hoja correspondiente y por ultimo agrega al reporte gramatical

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])")

reconoce la asignación de un valor a una variable entonces crea el nodo encargado de la asignación y el reporte gramatical

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])")

reconoce muchos corchetes izquierdos valor y corchetes derechos, agrega a la lista la expresión correspondiente y al reporte gramatical la accion

def p\_arreglo\_indice\_epsilon(t):  
 *'arra : '* t[0] = []  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('arra : ', "t[0] = []")

producción de terminación de los corchetes, sintetiza la lista de valores

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])")

reconoce la instrucción goto y crea la hoja correspondiente y por ultimo agrega al reporte gramatical

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])")

reconoce la instrucción ify crea la hoja correspondiente y por ultimo agrega al reporte gramatical

# 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])")

Crea el nodo de una operación que posee nodo izquierdo y derecho y agrega al reporte gramatical

def p\_expresion\_sola(t):  
 *'expresion : valor'* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('expresion : valor',"t[0] = t[1]")

expresión que se encarga de dar el valor si solo hay un valor sin operaciones  
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])")

Crea el nodo de una operación que posee nodo izquierdo y agrega al reporte gramatical  
  
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])")

Se encarga de los cast de los valores

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]")  
  
define la agrupación por los parentesis

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)")

define un entero

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)")

define un decimal

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)")  
  
define una cadena

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])")

define el valor de una variable

**Producciones Diferentes Del Analizador Descendente**

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] ")

solamente traslada la etiqueta

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])")

detecta una lista de etiquetas de manera recursiva por la derecha y agrega el valor usando la pila

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]])")

detecta la terminación de la recursividad y guarda la ultima etiqueta almacenada la pila

def p\_instrucciones\_lista(t):  
 *'instrucciones : instruccionesp '* t[0] = t[1]  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('instrucciones : instruccionesp ', "t[0] = t[1]")

pasa las instrucciones de la lista

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])")

almacena en la lista la ultima instrucción reconocida

def p\_instrucciones\_lista\_sin\_recursividad\_epsilon(t):  
 *'instruccionesp : '* t[0] = ABCInstruccion.ListaInstruccion([])  
  
 global Ts  
 Ts.rp\_nuevo\_nodo('instruccionesp : ', "t[0] = ABCInstruccion.ListaInstruccion([])")

inicializa la lista de instrucciones