Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de lenguajes y compiladores 1

Laboratorio Sección N

Tutor Académico: José Puac

Manual Técnico

Nombre: Byron Rubén Hernández de León

Carnet: 201806840

4 de julio de 2021

Introducción

El proyecto consiste en la elaboración de un intérprete de un lenguaje especificado previamente. El proyecto fue nombrado JPR según lo establecido, como objetivo de este documento tenemos el desglose de detalles de los métodos y lógica utilizada en cada ámbito.

Entorno de Trabajo

**Editor**

* Tkinter
* Filedialgos
* Webrowser
* OS
* Simpledialog
* scrolledText
* MesageDailod

**Diccionario**

* Análisis léxico: es el análisis que revisa si una palabra está bien escrita.
* Análisis sintáctico: es el que revisa el orden de las palabras.
* Análisis semántico: es el que revisa si el ámbito de las palabras está correcto.
* Dot: es un tipo de lenguaje utilizado por graphviz.
* Graphviz: es una librería para la generación de graficas.
* Nodo: es algo que describe un evento o dato, que forma parte un árbol.
* Árbol sintáctico: es una estructura de nodos generada por los pasos en la gramática.

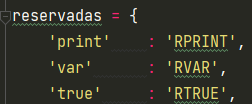
**Lenguaje JPR**

Para la realización del interprete requerimos de 3 análisis principales que son el léxico sintáctico y semántico, utilizamos un tipo de gramática ascendente ya que Utilizamos el tipo de gramática LALR la cual es la que maneja la librería yac y ply en su conjunto.

**Análisis Léxico**

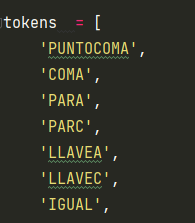
Para el análisis léxico utilizamos la modalidad de reservadas y tokens, en conjunto con Expresiones regulares para el análisis de las mismas.

* Reservadas



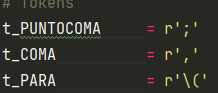
La lista de reservadas se declara como {llave,valor} lo que nos permitirá una fácil acceso

* Tokens



Los tokens primeros se les declara el nombre luego de eso tenemos que declararlos ya sea en una expresión regular o solo como una unidad léxica

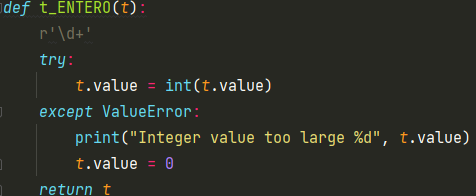
* Declaración por unidad lexic



* Declaración por expresión regular



* Declaración por función

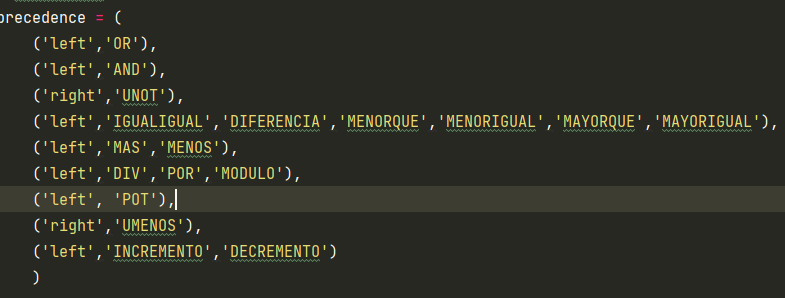
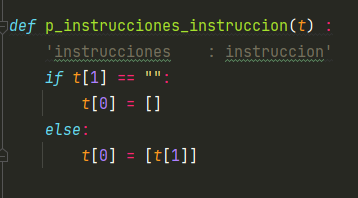
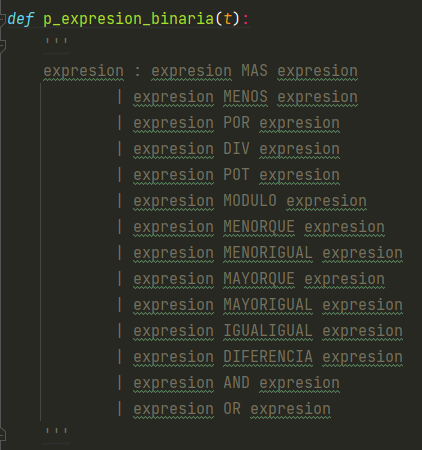
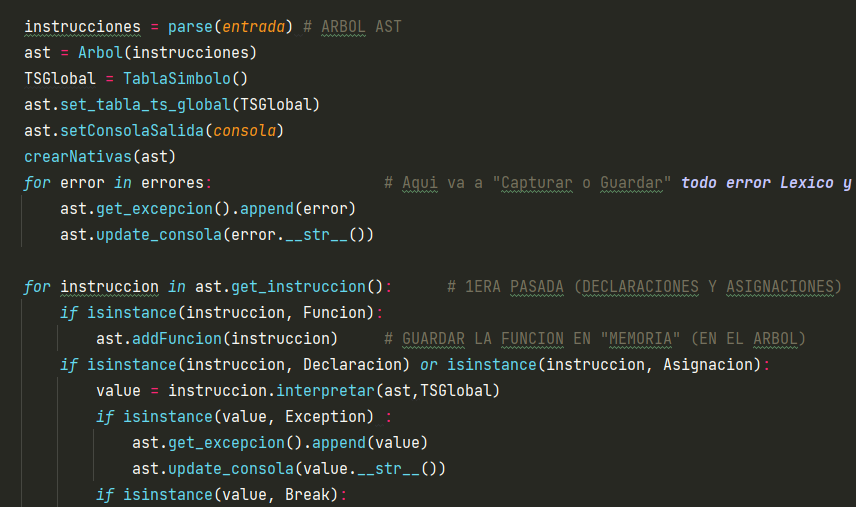
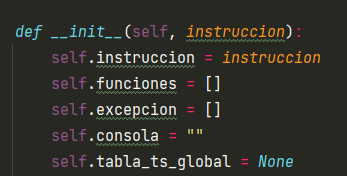
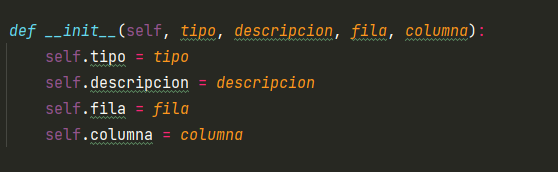
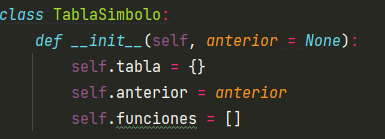
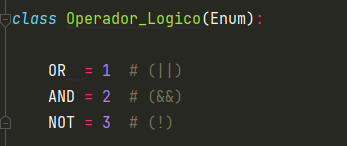
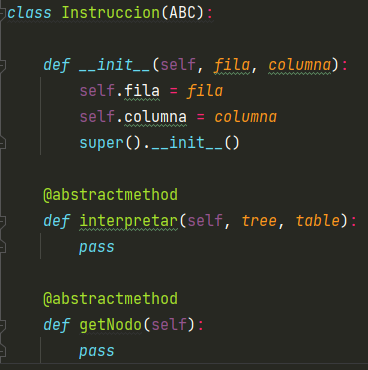
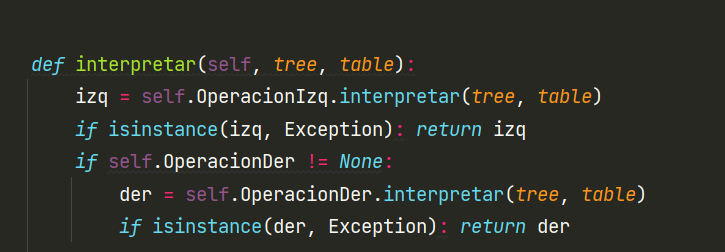
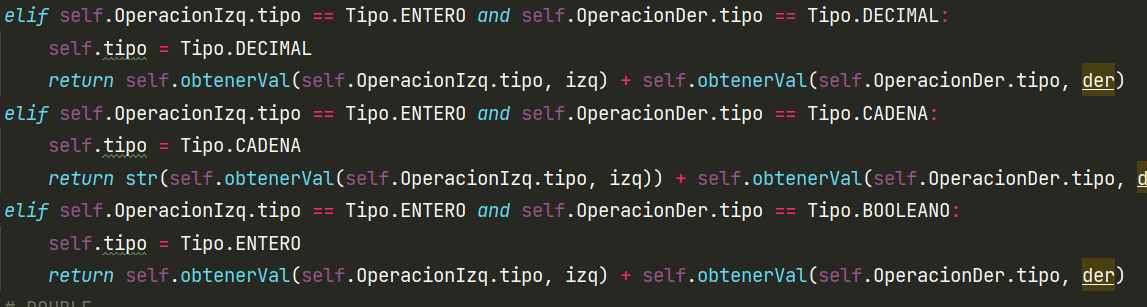
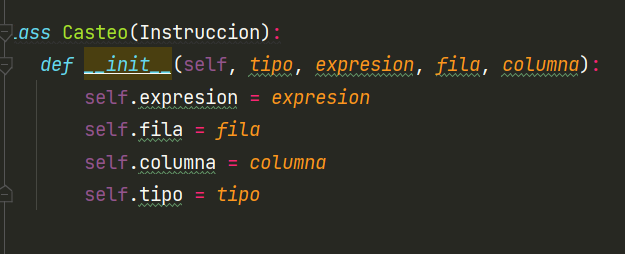
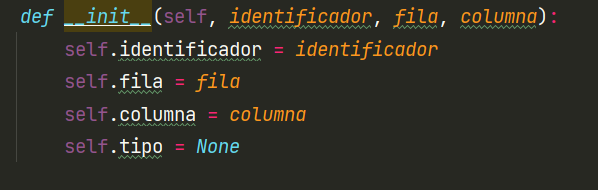
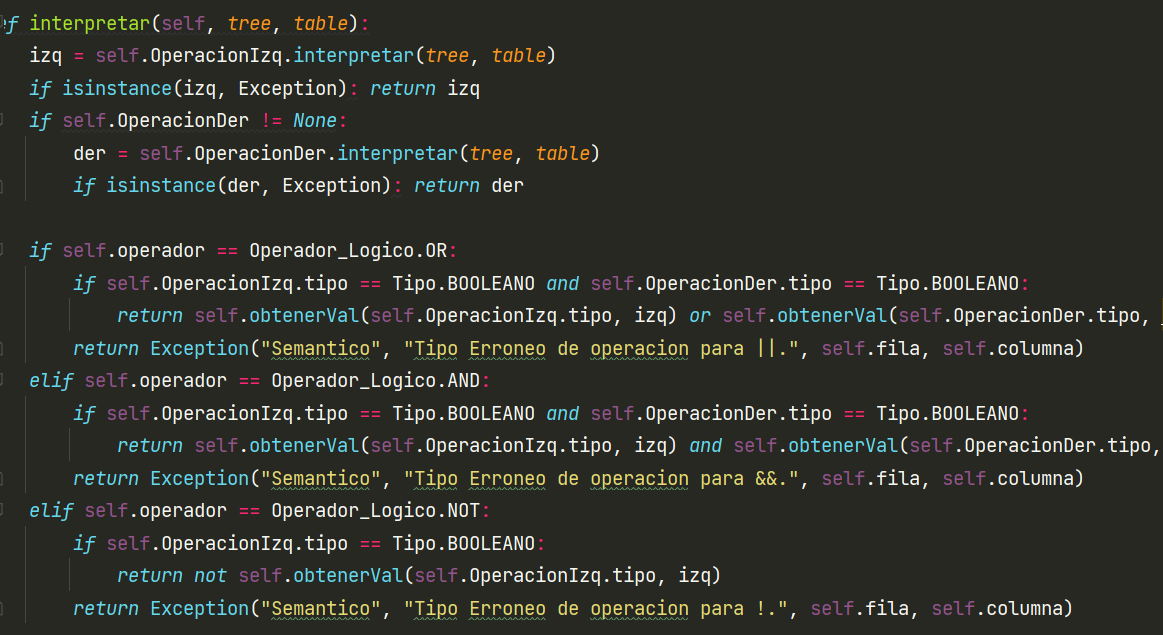
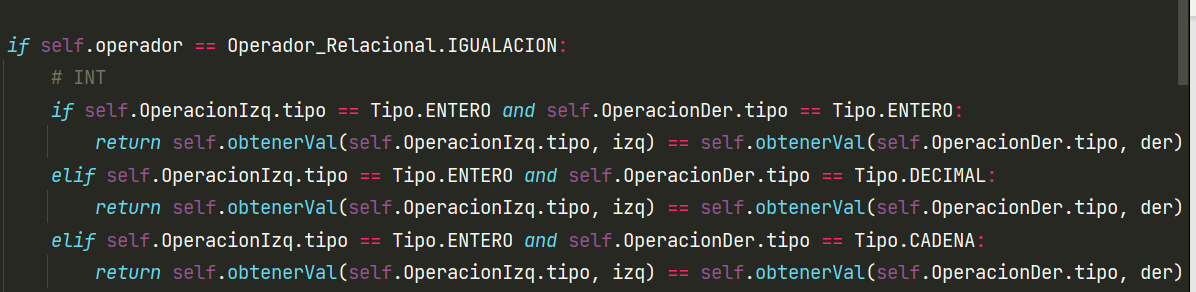
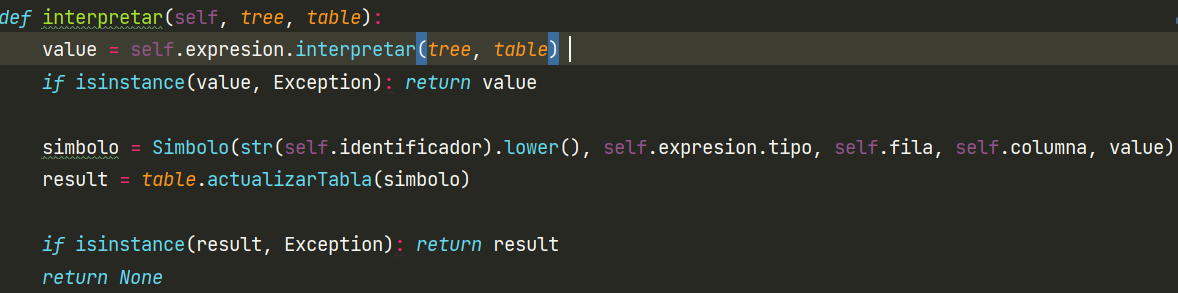
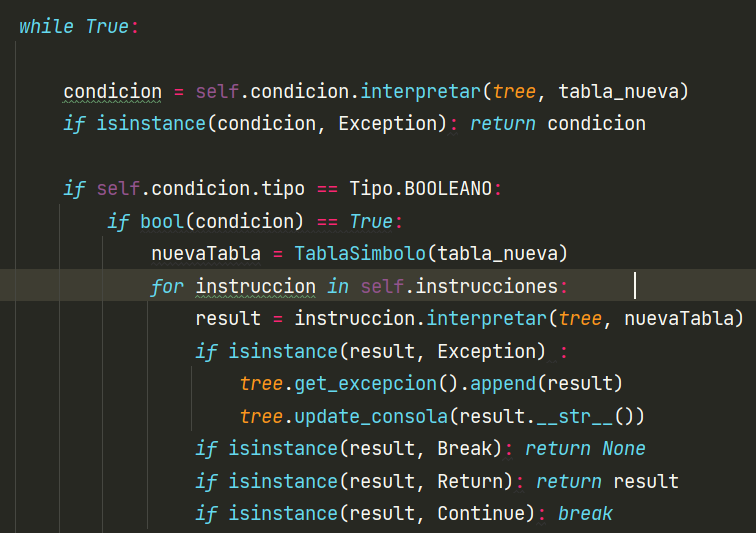


Las expresiones regulares son las siguientes

* t\_DECIMAL = ‘\d+.\d+’ expresion regular que evalua si el token es decimal, pudiendo venir cualquier digito una o mas veces, seguido de punto, seguido de digito una o mas veces.
* t \_CADENA = r"""\"(\\"|\\'|\\\\|\\n|\\t|\\r|[^\\\'\"])\*?\"""" expresion que evalua si el token es una cadena. En el lenguaje eta permitido colocar algo como lo siguiente: “\\” por lo que esta expresion evalua si solamente viene el simbolo “\” solo, entonces la cadena reconocida seria un error
* t\_CARACTER = r""" \'(\\'|\\\\|\\n|\\t|\\r|\\"|.)?\'""" Expresion que evalua si el token es un carácter, igualmente que con la cadena se evalua si el simbolo “\” viene solo. Recalcando para las cadenas y caracteres, se hizo un reemplazo en el token para reemplazar los simbolos de escape del lenguaje JPR ya que en el texto del token un “\\n” se definiria como tal y no como un salto de linea.
* t\_COMMENTLINES = r'\#\\*(.|\n)\*?\\*\#' Expresion que evalua si el token es un comentario multilinea.
* t\_COMMENTUNLINE = r'\#.\*\n' Expresion que evalua si es un comentario unilinea.

**Análisis sintáctico**

Para la realización del análisis sintáctico utilizamos una gramática recursiva ascendente por la izquierda debido a que nuestra herramienta utiliza el método de LALR para la realización del árbol sintáctico.

* Precedencia: se uso la precedencia en los operadores para poder tener una buena jerarquía de operaciones para la evaluación correcta de expresiones.  
  
* Gramática: se definió la gramática para poder generar las tablas de LALR de manera correcta para esto se utilizo varios tipos, pero una generalización seria la siguiente.  
  
* Gramática de Expresión Binaria: se utilizó para las comparaciones de operadores lógicos y aritméticos.  
  
* Método de Interprete: aquí es donde se junta todo lo declarado anteriormente y se genera un árbol del sintáctico.  
  
* Árbol: es una clase donde se crea y almacena el árbol por método de nodos para así poder tener un control de todo lo que se quiere.  
  
* Excepción: en esta clase se cran todos lo errores léxicos, sintácticos y semánticos que genera nuestro interprete en caso de una mala entrada.  
  
* Tabla Símbolo: en esta clase usamos nuestra clase símbolo que nos servirá para guardar todo lo declarado en los varios ámbitos.  
  
* Clase Enum: en esta clase declaramos todos los tipos de datos que pueden venir , ya sean operadores lógicos o datos primitivos.  
  
* Carpeta Abstract: en esta carpeta tenemos todos los métodos que usamos en cada clase desde instrucciones hasta expresiones, entre estos esta nodo,debugger e instrucción.  
   
* Carpeta Expresiones: en esta carpeta hay clases de todas las posibles variaciones que pueda tener un expresion y todo lo que abarca la palabra en conjunto.
* Aritmetica: en esta clase es donde evaluamos las operaciones aritemticas y que sus tipos sean permitidos.  
    
  
* Clase Casteo: en esta clase estan definidos todos los casteos para el interprete.  
  
* Clase identificador: en esta clase es donde creamos los id que reconoce el interprete  
  
* LOGICA: en esta clase interpretamos todos lo operadores logicos dentro de las expresiones.  
  
* RELACIONAL:En esta clase interpretamos todos los operadores realcionales en la expresion  
  
* Carpte Instrucciones: es donde almacenamos todas las intrucciones , bulces y sentencias para el interprete.
* Asignacion: es la instrucción de asignar el valor a un id indicado en una expresion.  
  
* FOR: es la intruccion de un bucle con una sentencia para que se detenga.  
  
* Metodo getNodo: este metodo se encuentra ubicado en todas las clases y es usado para poder obtener los datos suficiente para la generacion de la cadena del dot.  
  