Manual Técnico

Se le presenta el manual técnico de la parte 2 de la practica del curso de lenguajes formales y de programación, esta segunda parte consta de formar un analizador sintáctico. El lenguaje de programación Python es ampliamente reconocido por su simplicidad y facilidad de uso, lo que lo convierte en una opción popular entre desarrolladores de software. Sin embargo, para ejecutar programas escritos en Python de manera efectiva, es esencial contar con un analizador sintáctico que pueda comprender y validar la estructura de los códigos fuente.

Este proyecto representa la segunda fase de nuestro esfuerzo en el desarrollo de un analizador completo para el lenguaje Python. La primera parte se centró en la creación de un analizador léxico, encargado de descomponer el código fuente en "tokens" o unidades léxicas básicas. Ahora, en esta segunda etapa, nos sumergimos en la creación de un analizador sintáctico desde cero, sin depender de herramientas de creación sintáctica automatizadas como JCU.

Este enfoque manual nos permite un mayor control y comprensión de la estructura interna de Python, lo que es esencial para depurar y expandir nuestro analizador a medida que avanzamos. A través de este proyecto, exploraremos la complejidad del análisis sintáctico de Python, abordando la construcción de árboles de análisis y la validación de la sintaxis de manera rigurosa.

A medida que avanzamos en este proyecto, contribuimos al desarrollo de herramientas esenciales para la comunidad de programadores de Python. Nuestro objetivo es proporcionar un analizador sintáctico confiable y versátil que pueda utilizarse en diversos contextos, desde la depuración de código hasta la generación de documentación.

¡Únete a nosotros en este emocionante viaje a través del análisis sintáctico de Python y descubre cómo construir un analizador robusto sin depender de herramientas automatizadas!

Analizador Léxico:
El proceso comienza con el analizador léxico, que se encarga de dividir el código fuente en tokens. Cada token representa una unidad léxica básica del lenguaje, como palabras clave, identificadores, operadores y literales. Estos tokens se envían al analizador sintáctico para su procesamiento.

Analizador Léxico (Lexer) con JFlex

El Analizador Léxico, también conocido como Lexer, representa la primera etapa de nuestro proceso de análisis de código Python. Su principal misión es escanear el código fuente y segmentarlo en unidades léxicas fundamentales denominadas "tokens". Para esta tarea, hemos aprovechado JFlex, una herramienta ampliamente reconocida para la generación de analizadores léxicos.

Funciones Cruciales del Analizador Léxico:

Tokenización: El Analizador Léxico opera siguiendo reglas predefinidas para identificar patrones dentro del código fuente, tales como palabras clave, identificadores, números, cadenas, operadores y otros elementos léxicos. Cada vez que localiza uno de estos patrones, genera el token correspondiente, el cual será posteriormente procesado por el Analizador Sintáctico.

Eliminación de Espacios en Blanco y Comentarios: El proceso del Analizador Léxico también incluye la eliminación de espacios en blanco, tabulaciones y comentarios presentes en el código fuente. Esto simplifica el trabajo del Analizador Sintáctico, ya que no necesita abordar elementos de formato irrelevantes.

Generación de Tokens: Cada token producido por el Analizador Léxico se compone de dos componentes clave: el tipo de token (ejemplo: "ID" para identificadores o "INT" para números enteros) y el valor del token, que representa el texto específico que coincide con el patrón hallado en el código fuente. Estos tokens se almacenan en una estructura de datos apropiada para su posterior procesamiento.

Manejo de Palabras Clave y Símbolos Especiales: El Analizador Léxico identifica palabras clave propias del lenguaje Python, tales como if, while, def, así como símbolos especiales como operadores aritméticos y de asignación. A cada uno de ellos se le asigna el tipo de token correspondiente.

Detección de Errores Léxicos: La detección de errores léxicos, como caracteres inválidos o secuencias incorrectas, también forma parte de las responsabilidades del Analizador Léxico. Cuando surge un error, se genera un mensaje de error detallado, brindando una guía útil para que los desarrolladores identifiquen y corrijan los problemas en el código.

Ventajas de Utilizar JFlex:

La elección de JFlex para implementar el Analizador Léxico conlleva múltiples ventajas, entre las cuales destacan:

Definiciones basadas en Expresiones Regulares: JFlex permite la definición de patrones de reconocimiento de tokens mediante expresiones regulares, lo que simplifica la especificación de las reglas léxicas del lenguaje.

Generación Automatizada de Código: JFlex automatiza la generación del código Java correspondiente al Analizador Léxico a partir de las definiciones proporcionadas, lo que ahorra tiempo y disminuye la probabilidad de errores humanos.

Eficiencia en Rendimiento y Recursos: Los Analizadores Léxicos generados por JFlex se destacan por su eficiencia en cuanto a velocidad y uso de recursos, características cruciales para el análisis ágil del código fuente.

En resumen, el Analizador Léxico desarrollado con JFlex se erige como un componente crítico en nuestro sistema de análisis de código Python. Su función primordial consiste en desglosar el código fuente en tokens, permitiendo que el Analizador Sintáctico proceda con la validación y análisis subsiguientes.

```
"-" |
"**" |
"/" |
"//" |
"%" |
"*" |
"+" {posicion = linea; lexeme =yytext(); return Aritmeticos;}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}

"&" |
"|" |
"^" |
"<<" |
">>" {posicion = linea; lexeme =yytext(); return Bits;}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}

"==" |
"!=" |
">" |
"<" |
">=" |
"<=" {posicion = linea; lexeme =yytext(); return Comparacion;}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}
```

Parser o Analizador Sintáctico:
El núcleo del analizador sintáctico es el parser, que utiliza una gramática formal que describe la estructura de Python. Este componente procesa los tokens proporcionados por el analizador léxico y construye un árbol de análisis sintáctico. Este árbol representa la estructura jerárquica del programa, incluyendo las relaciones entre instrucciones, bloques, expresiones y otros elementos del lenguaje.
2.2 Analizador Sintáctico (Parser) con Tabla de Símbolos y Generación de Tablas y Reportes

El Analizador Sintáctico es la pieza central de nuestro sistema de análisis de código Python. Su tarea principal es verificar si el código fuente cumple con las reglas de sintaxis del lenguaje, utilizando una gramática formal predefinida. Además, este componente opera en conjunto con otros módulos esenciales que enriquecen su funcionalidad. A continuación, se presentan las partes clave de este analizador:

Tabla de Símbolos:

La Tabla de Símbolos desempeña un papel fundamental en el análisis sintáctico. Esta estructura de datos mantiene un registro de variables, funciones y otros elementos definidos en el programa. Durante el análisis, el Analizador Léxico agrega información a la tabla de símbolos, lo que permite realizar comprobaciones de uso de variables, resolución de nombres y análisis semántico. La tabla de símbolos facilita la identificación de elementos en el código y su referencia en el proceso de análisis.

Clases para Creación de Reglas y Verificación de Cumplimiento:

En esta fase del proyecto, hemos implementado varias clases que se encargan de definir y verificar reglas de sintaxis y cumplimiento. Estas clases representan la gramática formal del lenguaje Python y determinan cómo se deben estructurar las construcciones del código. Además, se utilizan para garantizar que las reglas sintácticas se cumplan rigurosamente durante el análisis. Estas clases son esenciales para la construcción del árbol de análisis sintáctico y para la generación de tablas y reportes detallados.

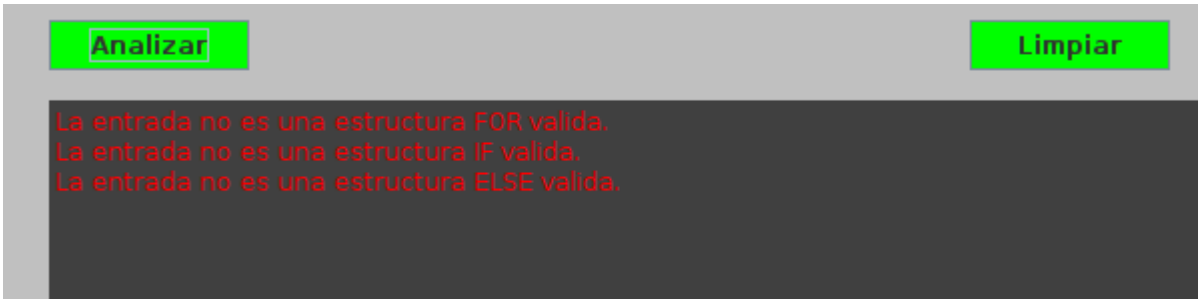
Clases para la Creación de Tablas y Reportes:

El Analizador Sintáctico también incluye clases adicionales que se encargan de generar tablas y reportes para resumir la estructura del programa analizado. Estos informes pueden incluir detalles sobre variables definidas, funciones, clases y otros elementos del código. La generación de tablas y reportes es esencial para fines de documentación, análisis estático y optimización de código. Estos recursos ayudan a los desarrolladores a comprender y depurar su código de manera más efectiva.

En conjunto, el Analizador Sintáctico, la Tabla de Símbolos y las clases para la creación de reglas y reportes conforman un sistema integral que verifica y analiza el código fuente Python en profundidad. Este proceso contribuye a la identificación de errores sintácticos, la creación de una estructura interna representada como un árbol de análisis sintáctico y la generación de documentación y reportes que facilitan la comprensión y mejora del código fuente.

```java
public class Analizador {
    public static boolean analizarFor(String[] entrada) {
        String regexFor = "for\\s+[a-zA-Z_][a-zA-Z_0-9]*\\s+in\\s+range\\s*\\(\\s*\\d+\\s*\\)\\s*:";

        Pattern patron = Pattern.compile(regex: regexFor);

        // Concatenar todos los elementos en una sola cadena
        StringBuilder cadena = new StringBuilder();
        for (String elemento : entrada) {
            cadena.append(str:elemento).append(str:" ");
        }

        String cadenaCompleta = cadena.toString().trim();

        // Verificar si la cadena cumple con el patrón al inicio o en cualquier posición
        Matcher matcher = patron.matcher(input: cadenaCompleta);

        return matcher.matches() || cadenaCompleta.matches("^.*" + regexFor + ".*$");
    }
```

Ejemplo de creacion de regla para analizar el ciclo For.



Manejo de errores.

Tabla de Símbolos:
La tabla de símbolos es una estructura de datos esencial para el seguimiento de variables, funciones y otros elementos definidos en el programa. El analizador sintáctico agrega información a esta tabla a medida que analiza el código. Esto permite realizar comprobaciones de uso de variables, resolución de nombres y análisis semántico.

Manejo de Errores:
El analizador sintáctico debe ser capaz de detectar errores sintácticos en el código, como estructuras mal formadas o elementos faltantes. Además, se deben implementar mecanismos para generar mensajes de error claros y, en algunos casos, recuperarse de errores para continuar analizando el código.

Generación de Tablas y Reportes:
Como parte de su función de análisis, el analizador sintáctico puede generar tablas y reportes que resumen la estructura del programa analizado. Estos informes pueden incluir detalles sobre variables definidas, funciones, clases y otros elementos del código, lo que resulta útil para documentación, análisis estático y optimizaciones de código.

Esta estructura del analizador sintáctico se integra en un entorno más amplio de desarrollo de software, donde puede interactuar con otros componentes, como el generador de código intermedio y el optimizador, si es necesario. El objetivo principal de este analizador es asegurar que el código Python sea sintácticamente válido y estructuralmente coherente para su posterior procesamiento o ejecución.