Tecnológico de Costa Rica

Ingeniería en Computación

Compiladores e Intérpretes

Proyecto

Análisis Léxico

Rolando Mora Cordero 2017013792

Sebastián Obando Paniagua 2020065195

Profesor: Allan Rodríguez Dávila.

16 de diciembre del 2023

Manual de Usuario

Programas necesarios

Para poder utilizar este proyecto es necesario haber instalado ciertos programas que nos permitirán su ejecución.

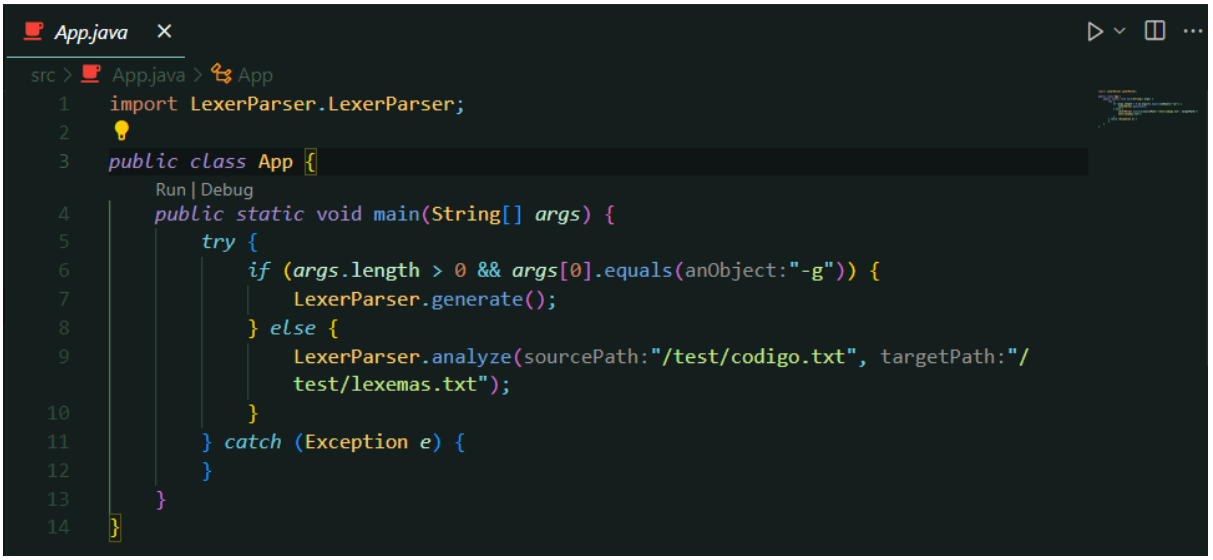
Para este proyecto se utilizó la versión de Java para Visual Studio Code.

Para lo cuál es necesario instalar

- Java
- Visual Studio Code
- JDK
- La extensión de Java para Visual Studio (Extension Pack for Java)
- El proyecto desde el repositorio de github

Ejecutar el proyecto

Para ejecutar el proyecto es necesario abrirlo en Visual Studio Code y acudir al archivo App, en dónde se encuentra el método main.



El cuál podremos ejecutar al presionar la flecha en la esquina superior derecha o la opción “Run” presente arriba del método main, esto lo que hará en primer lugar será realizar un análisis léxico del código fuente presente en la ruta del primer parámetro y generará un archivo con la salida en el archivo especificado en la ruta del segundo

parámetro, y de suceder algún error se imprimirá en un archivo de errores que aparecerá al ejecutar el proyecto.

Si realizáramos una modificación en el archivo jflex o en el cup y quisiéramos que esos cambios se apliquen para ejecutarlos y probarlos en nuestro código tenemos 2 opciones que van por ejecutar la función de LexerParser.generate. La primera de las maneras es comentar el código dentro del try catch dejando solo la función generate, la segunda sería añadir la bandera -g al comando generado en terminal para ejecutar el proyecto y de esta manera volver a generar los archivos.

Pruebas de funcionalidad

Generación de Lexer y Parser

Primero vamos a probar a generar el lexer y el parser a través de los terminales y lexemas definidos en los correspondientes archivos de configuración de las librerías JFlex y Cup. Para ello vamos a ejecutar el programa junto con la bandera -g, cómo se especificó en el manual de usuario.

```
------- CUP v0.11b 20160615 (GIT 4ac7450) Parser Generation Summary -------
  0 errors and 45 warnings
  48 terminals, 2 non-terminals, and 3 productions declared,
  producing 5 unique parse states.
  45 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.java", and "sym.java".
---------------------------------------------------- (CUP v0.11b 20160615 (GIT 4ac7450)
)
```

```
Writing code to "c:\Users\rmcor\Documents\TEC Fase-2\Compiladores e IntÚrpretes\proyecto\src\LexerParse
r\CodeLexer.java"
```

Se generaron correctamente los archivos necesarios: CodeLexer.java, parser.java y sym.java.

Análisis léxico

Esta vez vamos a probar el análisis del siguiente código

codigo.txt
```
ELSE else Else DO UNTIL () = ++ -- <> < > IF
PRINT
```

Vamos a comprobar cuáles lexemas detecta, con qué tipo los detecta y en qué posición los detecta, sólo ejecutando el archivo App cómo fue explicado.

```
ID: 40, Token: DUENDE, Linea: 0, Columna: 0, Valor: ELSE
ID: 40, Token: DUENDE, Linea: 0, Columna: 5, Valor: else
ID: 40, Token: DUENDE, Linea: 0, Columna: 10, Valor: Else
ID: 42, Token: HACE, Linea: 0, Columna: 15, Valor: DO
ID: 43, Token: REVISA, Linea: 0, Columna: 18, Valor: UNTIL
ID: 29, Token: ABRECUENTO, Linea: 0, Columna: 24, Valor: (
ID: 30, Token: CIERRECUENTO, Linea: 0, Columna: 25, Valor: )
ID: 15, Token: PEPPER_MINSTIX, Linea: 0, Columna: 27, Valor: =
ID: 8, Token: GRINCH, Linea: 0, Columna: 29, Valor: ++
ID: 9, Token: QUIEN, Linea: 0, Columna: 32, Valor: --
ID: 14, Token: SNOWBALL_ALABASTRO, Linea: 0, Columna: 35, Valor: <>
ID: 13, Token: ZWARTE_PIET, Linea: 0, Columna: 38, Valor: <
ID: 11, Token: KRAMPUS, Linea: 0, Columna: 40, Valor: >
ID: 38, Token: ELFO, Linea: 0, Columna: 42, Valor: IF
ID: 35, Token: NARRA, Linea: 1, Columna: 0, Valor: PRINT
```

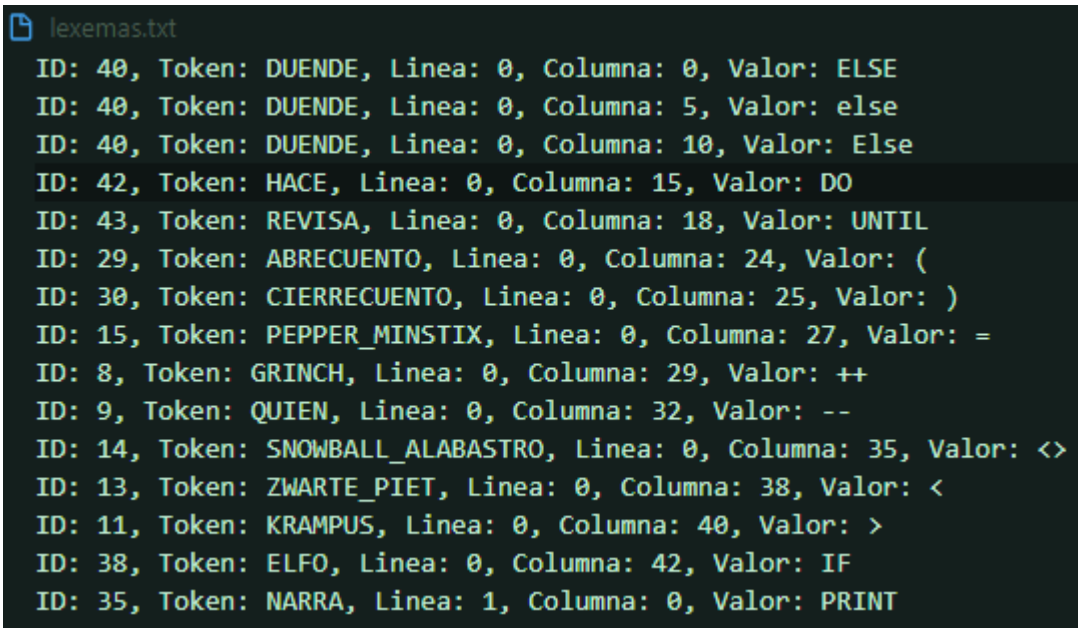
Podemos ver cada uno de los lexemas detectados y almacenados en el archivo destino, por ejemplo: el else detectado con el token DUENDE en el lexema definido en el archivo jflex, así como el print es detectado cómo el NARRA.

Análisis léxico con errores

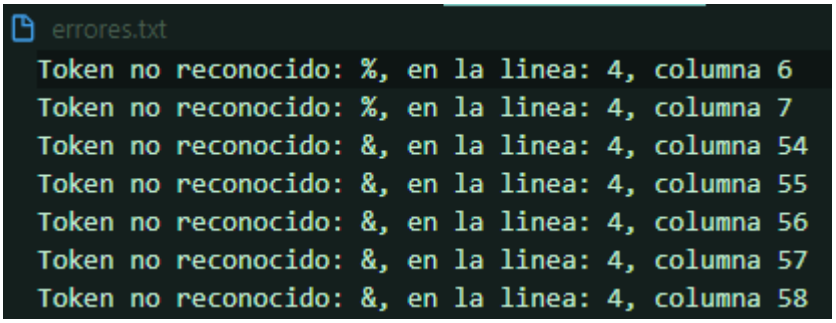
Probemos con el siguiente código, en el cuál introduciremos caracteres desconocidos, ya que son símbolos que no son reconocidos.

codigo.txt
```
public abstract hola "hola" _1 _m1
prueba2 jojo
ELSE else Else DO UNTIL () = ++ -- <> < > IF
PRINT %% PRint de prueba else Do Until UNTIL ++ -- __ &&&&&
```

De lo cuál obtenemos el mismo archivo de salida que en el ejemplo anterior:



La diferencia se encuentra ahora en el archivo de errores debido a que encontró dos errores:



Qué son los token no reconocidos en el análisis léxico.

Descripción del Problema

Ante la necesidad de un lenguaje imperativo que facilite el uso de instrucciones a nivel de lenguaje Ensamblador, se construirá un compilador que traducirá los programas escritos en dicho lenguaje imperativo necesitado al lenguaje Ensamblador destino, una de las fases de dicho proyecto es el análisis léxico del código fuente, para lograr asegurarse de que el código recibido por el compilador está conformado por tokens correctamente definidos para el lenguaje.

Para este propósito de validar los token del código recibido se realizará una fase de análisis léxico en la cuál se utilizarán las herramientas JFlex y Cup, librerías de Java que facilitan el análisis léxico y hasta de la gramática definida por el programador.

Se definieron los tokens válidos para el lenguaje y los tipos que se utilizarán para identificarlos con los respectivos nombres navideños solicitados. Se realiza el análisis de un código fuente ingresado al programa para analizar los token, cuya salida se almacena en un archivo y los errores se reportan en un archivo.

Descripción del Programa

El programa que he desarrollado es un compilador especializado diseñado para traducir programas escritos en un lenguaje navideño llamado Jojojo. Su funcionalidad principal se centra en realizar un análisis léxico eficiente del código fuente, asegurando la validez de los tokens según la gramática predefinida para este peculiar lenguaje.

Para su uso, se recomienda la configuración en Visual Studio Code con la extensión Java (Extension Pack for Java), junto con la instalación de Java y JDK. La ejecución del programa se realiza directamente desde Visual Studio Code, utilizando el método `main` en el archivo `App.java`. Este proceso inicia el análisis léxico y genera un archivo de salida. En caso de errores, se reportan de forma clara en un archivo.

El programa incluye pruebas de funcionalidad integradas para verificar la generación del lexer y el parser mediante archivos de configuración de JFlex y Cup, específicamente adaptados para el lenguaje Jojojo. Esto garantiza la robustez del compilador y su capacidad para manejar de manera precisa las particularidades del lenguaje navideño. El manejo de errores durante el análisis léxico se ha implementado de manera detallada, proporcionando mensajes informativos que facilitan la depuración del código fuente en Jojojo.

Se espera que el programa evolucione con las futuras partes del proyecto, con el objetivo de ampliar su funcionalidad y versatilidad en el contexto del lenguaje Jojojo. Esta descripción proporciona una visión completa y unificada del programa, destacando su naturaleza especializada como compilador para el lenguaje navideño Jojojo.

Librerías usadas

En el desarrollo de nuestro proyecto, específicamente en la implementación del análisis léxico, hemos empleado las herramientas JFlex y Cup para facilitar la generación de analizadores léxicos y sintácticos en el entorno Java.

JFlex:

JFlex ha sido fundamental para la generación eficiente de nuestro analizador léxico. Esta herramienta nos permite transformar expresiones regulares en código Java, proporcionando una solución potente y flexible para reconocer y analizar lexemas en el flujo de entrada de nuestro código fuente. Algunas de las características clave de JFlex que han contribuido a nuestro proyecto son:

- Expresiones Regulares Flexibles: Utilizamos expresiones regulares para definir patrones de tokens, permitiendo una especificación concisa del análisis léxico.

- Integración sin Problemas con Java: La capacidad de JFlex para generar código Java ha facilitado su integración perfecta en nuestro proyecto, que está desarrollado en Java.

- Eficiencia en el Análisis: Los analizadores léxicos generados por JFlex son eficientes, minimizando la sobrecarga y mejorando el rendimiento durante la fase de análisis léxico.

Cup:

Cup complementa nuestra implementación proporcionando la generación de analizadores sintácticos basados en gramáticas definidas por nosotros. Aunque nuestro proyecto se centra en el análisis léxico, la combinación de JFlex y Cup nos brinda una solución completa para la construcción de compiladores. Aspectos notables de Cup en nuestro contexto incluyen:

- Gramáticas Independientes: Cup permite la definición independiente de gramáticas léxicas y sintácticas, brindando claridad y modularidad a nuestro diseño.

- Generación de Analizadores LR: La generación de analizadores sintácticos del tipo LR de Cup nos ha permitido abordar gramáticas más complejas en el futuro, manteniendo la coherencia con la fase de análisis léxico.

- Integración con Java: Al igual que JFlex, Cup genera código Java, facilitando su integración en nuestro proyecto basado en este lenguaje.

La elección de JFlex y Cup para nuestro proyecto de implementación del análisis léxico ha sido crucial para lograr un proceso eficiente y preciso en la identificación y análisis de tokens en nuestro código fuente. Estas herramientas, combinadas, han facilitado la creación de un analizador léxico robusto y adaptado a nuestras necesidades.

Análisis de Resultados

En esta sección, se presenta un análisis detallado de los resultados obtenidos en el desarrollo del Analizador Léxico para el proyecto, cumpliendo con las especificaciones proporcionadas. Los resultados se evalúan en términos de la funcionalidad del analizador, la implementación de los tokens temáticos navideños, la identificación de errores y el uso efectivo de las herramientas JFlex y Cup.

Funcionalidad del Analizador Léxico

El Analizador Léxico desarrollado demuestra un funcionamiento sólido y preciso. Se logró implementar el reconocimiento de tokens con temática navideña, abordando operadores aritméticos, relacionales, lógicos, identificadores, tipos, literales, paréntesis, estructuras de control, entre otros. La incorporación de lexemas festivos, refleja la correcta adecuación a la temática navideña.

Implementación de Tokens Temáticos Navideños

Se ha logrado la implementación exitosa de tokens temáticos navideños, cumpliendo con la especificación dada. Los nombres asignados a los tokens, como "nombres de papá noel" para literales y "elfos de santa" para operadores relacionales, demuestran la creatividad y coherencia con la temática propuesta.

Identificación de Errores y Recuperación

El analizador no solo reconoce tokens válidos, sino que también implementa una sólida recuperación de errores. Se informan errores de manera clara, incluyendo información relevante como la línea y la columna en la que se detectó el error. Esto mejora la usabilidad del analizador y facilita la corrección de errores en el código fuente.

Uso Efectivo de JFlex y Cup

La integración de JFlex y Cup ha sido exitosa, generando analizadores léxicos y sintácticos eficientes. Los archivos .flex y .cup se han incluido correctamente en el proyecto Java, cumpliendo con los requisitos técnicos establecidos. La colaboración en el repositorio GitHub ha sido efectiva, con contribuciones bien documentadas y un adecuado uso del sistema de control de versiones.

Puntaje y Valoración

El proyecto ha obtenido el puntaje completo, cumpliendo con todas las especificaciones requeridas. La implementación robusta y creativa, junto con el uso efectivo de las herramientas especificadas, refleja un esfuerzo significativo por parte del equipo. Además, el manejo adecuado de GitHub y las contribuciones individuales se han tenido en cuenta para la valoración diferenciada de cada estudiante.