



Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Compiladores e Intérpretes

Proyecto I | Análisis Léxico

Integrantes:

Casey Baeza Castrillo– Carné: 2022437750

Esteban Rojas Hidalgo – Carné: 2022078483

Profesor:

Allan Rodríguez Dávila

Verano 2023

Índice

Manual de usuario	3
Pruebas de funcionalidad	6
Descripción del problema	7
Diseño del programa	7
Librerías usadas	7
Análisis de resultados	9
Bitácora	9

Manual de usuario

- Requisitos Previos

Java Development Kit (JDK) instalado.

Apache Maven instalado y configurado correctamente.

JFlex y Cup instalado correctamente:

JFlex: <https://www.jflex.de/download.html>

Cup: <http://www2.cs.tum.edu/projects/cup/install.php>

Instrucciones:

1. Abrir proyecto

- a. Opción 1: Clonar repositorio

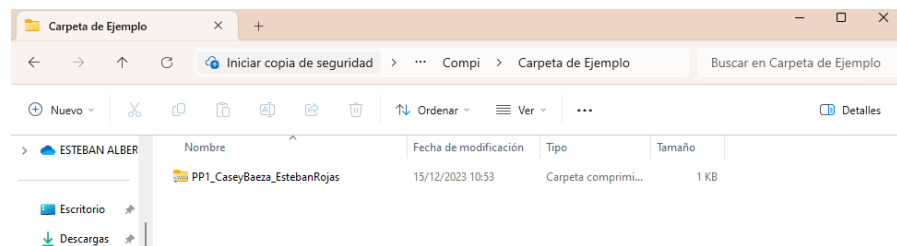
```
git clone https://github.com/cbaeza16/lexer.git
```

```
cd lexer
```

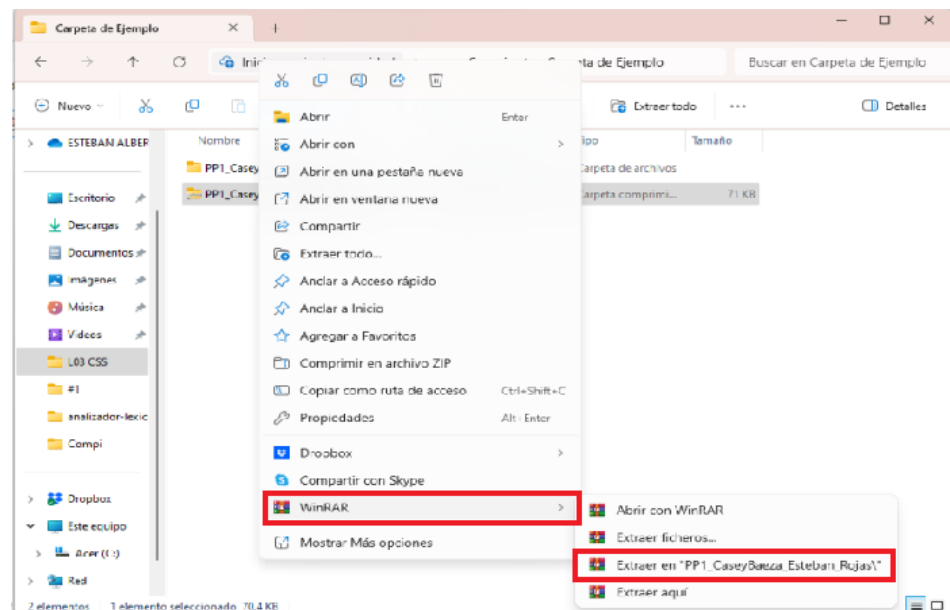
```
cd analizador-lexico
```

- b. Opción 2: Descargar .zip

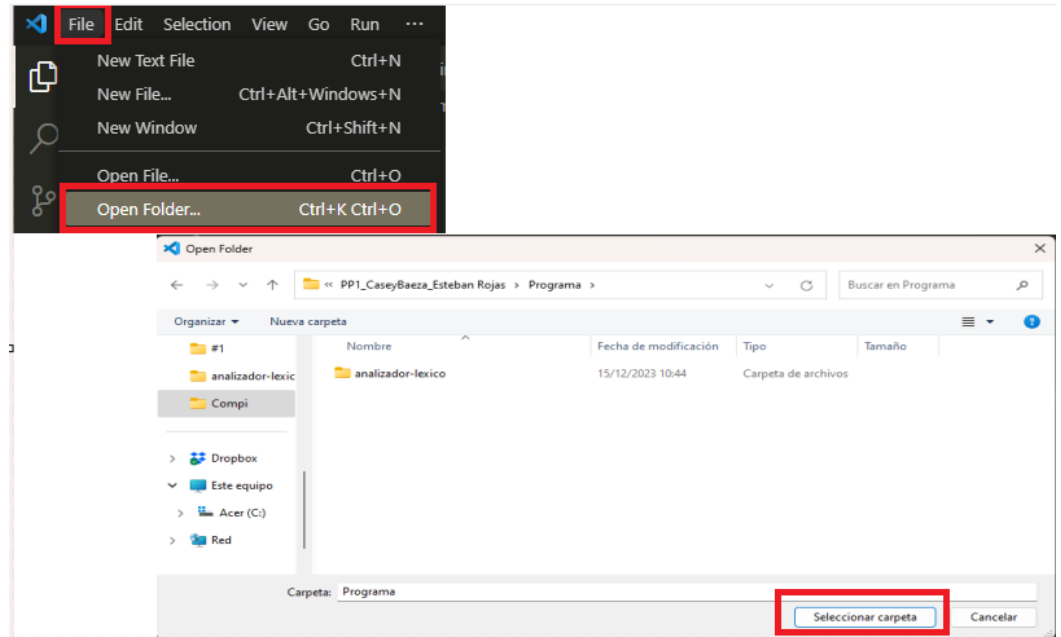
- i. Copiar el .zip en la carpeta destino deseada



- ii. Extraer los archivos en la carpeta seleccionada



- iii. Abrir el proyecto “analizador-lexico” ("PP1_CaseyBaeza_Esteban Rojas\Programa\analizador-lexico") desde el ambiente de su preferencia



2. Compilar proyecto

`mvn clean package`

3. Ejecución del proyecto

- a. Ejecutar el programa

`mvn exec:java`

Al usar este comando se ejecuta el programa (método main) en el cual se escanea el archivo fuente “text.txt”. En caso de querer utilizar otro archivo fuente, agregar archivo fuente bajo la ruta relativa “src/main\” y cambiar el nombre del archivo en “fileName” en método main (“src/main/java/com/p1/Main.java”).

- b. En caso de querer generar Analizador Léxico:

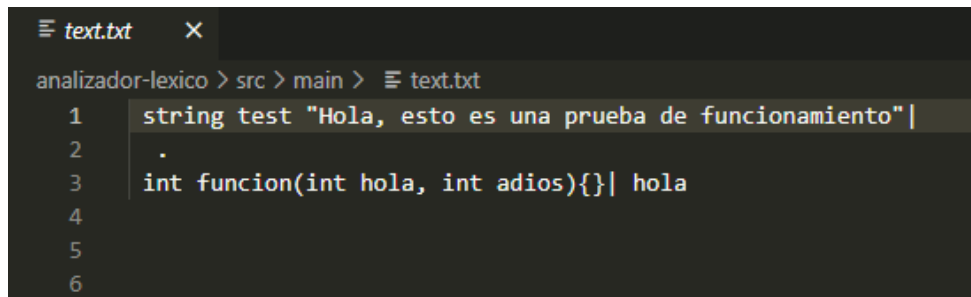
`mvn jflex:generate`

- c. En caso de querer generar Analizador Sintáctico:

`mvn cup:generate`

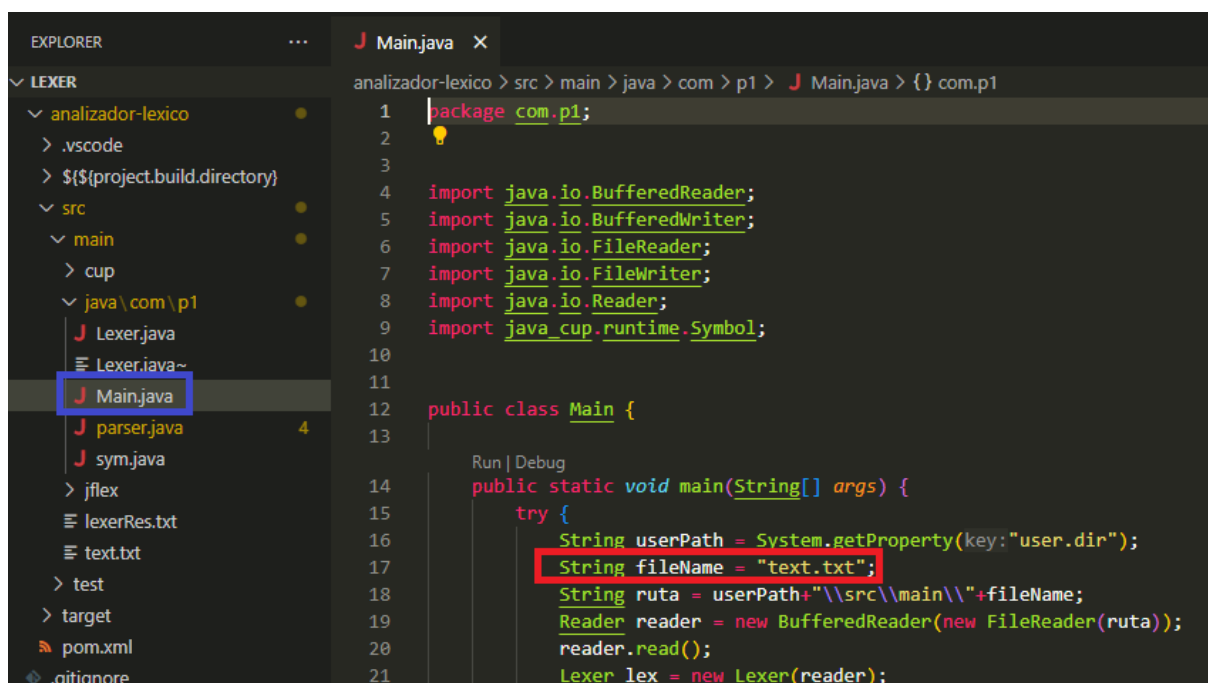
Pasos para su uso

Localizar o crear el archivo que se quiere analizar. Para este ejemplo se utiliza un archivo “text.txt”:



```
analizador-lexico > src > main > text.txt
1  string test "Hola, esto es una prueba de funcionamiento"|
2  .
3  int funcion(int hola, int adios){}| hola
4
5
6
```

Dentro del proyecto, en la clase Main.java, incluir el nombre del archivo a analizar.



```
EXPLORER
└─ LEXER
   └─ analizador-lexico
       └─ .vscode
       └─ ${project.build.directory}
       └─ src
           └─ cup
           └─ java\com\p1
               ├── Lexer.java
               ├── Lexer.java~
               └─ Main.java
                   ├── parser.java
                   └─ sym.java
               └─ jflex
               └─ lexerRes.txt
               └─ text.txt
               └─ test
               └─ target
               └─ pom.xml
               └─ .gitignore

Main.java
analizador-lexico > src > main > java > com > p1 > Main.java > {} com.p1
1  package com.p1;
2
3
4  import java.io.BufferedReader;
5  import java.io.BufferedWriter;
6  import java.io.FileReader;
7  import java.io.FileWriter;
8  import java.io.Reader;
9  import java_cup.runtime.Symbol;
10
11
12  public class Main {
13
14      Run | Debug
15      public static void main(String[] args) {
16          try {
17              String userPath = System.getProperty(key:"user.dir");
18              String fileName = "text.txt";
19              String ruta = userPath+"\\src\\main\\"+fileName;
20              Reader reader = new BufferedReader(new FileReader(ruta));
21              reader.read();
22              Lexer lex = new Lexer(reader);
23          }
24      }
25  }
```

Ya está listo para correr el programa. En la carpeta del proyecto se generará un archivo “lexerRex.txt” con los tokens encontrados y su respectiva línea y columna.

Pruebas de funcionalidad

Documento a analizar:

```
analizador-lexico > src > main > text.txt
1 string test "Hola, esto es una prueba de funcionamiento"|
2 .
3 int funcion(int hola, int adios){}| hola
```

Salida al correr el programa:

```
Run | Debug
14 public static void main(String[] args) {
15     try {
16         String userPath = System.getProperty(key:"user.dir");
17         String fileName = "text.txt";

Token: 20, Lexema: hola, Linea: 2, Columna: 37
Cantidad de lexemas encontrados: 17
PS C:\Users\... \lexer\analizador-lexico> c:; cd 'c:\Users\
zador-lexico'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users
e' 'com.pl.Main'
Token: 24, Lexema: string, Linea: 0, Columna: 0
Token: 20, Lexema: test, Linea: 0, Columna: 7
Token: 31, Lexema: "Hola, esto es una prueba de funcionamiento", Linea: 0, Columna: 55
Token: 51, Lexema: |, Linea: 0, Columna: 56
Error léxico en la línea 1 Columna: 2: Cadena ilegal <.>
Token: 22, Lexema: int, Linea: 2, Columna: 1
Token: 20, Lexema: funcion, Linea: 2, Columna: 5
Token: 33, Lexema: (, Linea: 2, Columna: 12
Token: 22, Lexema: int, Linea: 2, Columna: 13
Token: 20, Lexema: hola, Linea: 2, Columna: 17
Token: 48, Lexema: ,, Linea: 2, Columna: 21
Token: 22, Lexema: int, Linea: 2, Columna: 23
Token: 20, Lexema: adios, Linea: 2, Columna: 27
Token: 34, Lexema: ), Linea: 2, Columna: 32
Token: 37, Lexema: {, Linea: 2, Columna: 33
Token: 38, Lexema: }, Linea: 2, Columna: 34
Token: 51, Lexema: |, Linea: 2, Columna: 35
Token: 20, Lexema: hola, Linea: 2, Columna: 37
Cantidad de lexemas encontrados: 17
```

Documento generado:

```
EXPLORER
... J Main.java lexRes.txt J Lexer.java parser.cup lexer.jflex text.txt

ANALIZADOR-LEXICO
src > main > lexRes.txt
1 Token: 24, Lexema: string, Linea: 0, Columna: 0
2 Token: 20, Lexema: test, Linea: 0, Columna: 7
3 Token: 31, Lexema: "Hola, esto es una prueba de funcionamiento", Linea: 0, Columna: 55
4 Token: 51, Lexema: |, Linea: 0, Columna: 56
5 Token: 22, Lexema: int, Linea: 2, Columna: 1
6 Token: 20, Lexema: funcion, Linea: 2, Columna: 5
7 Token: 33, Lexema: (, Linea: 2, Columna: 12
8 Token: 22, Lexema: int, Linea: 2, Columna: 13
9 Token: 20, Lexema: hola, Linea: 2, Columna: 17
10 Token: 48, Lexema: ,, Linea: 2, Columna: 21
11 Token: 22, Lexema: int, Linea: 2, Columna: 23
12 Token: 20, Lexema: adios, Linea: 2, Columna: 27
13 Token: 34, Lexema: ), Linea: 2, Columna: 32
14 Token: 37, Lexema: {, Linea: 2, Columna: 33
15 Token: 38, Lexema: }, Linea: 2, Columna: 34
16 Token: 51, Lexema: |, Linea: 2, Columna: 35
17 Token: 20, Lexema: hola, Linea: 2, Columna: 37
18 Cantidad de lexemas encontrados: 17
```

Descripción del problema

Para este proyecto se debe diseñar un lenguaje imperativo especializado para la configuración de chips en sistemas empotrados, es fundamental que sea un lenguaje ligero y potente ya que es una industria en constante evolución, donde los chips requieren configuraciones cada vez más precisas y complejas.

El enfoque principal se basa en la fase de Análisis Léxico, donde se debe construir un scanner usando JFlex para reconocer tokens, identificar errores léxicos y aplicar una técnica de recuperación en Modo Pánico para asegurar que el análisis continúe a pesar de encontrar errores. Además, se establece la estructura básica del programa en este nuevo lenguaje, compuesto por declaraciones de procedimientos, expresiones y asignaciones.

Diseño del programa

En base a la estructura del proyecto, se utiliza una estructura Maven, para así tener directorios bien organizados y claramente definidos para código fuente, recursos, archivos generados, etc.

En cuanto a la implementación del Analizador Léxico, se utiliza JFlex para generar el analizador léxico basado en las reglas definidas en la gramática del lenguaje. Además, se utiliza CUP para generar el parser y sym.

Según la estructura e implementación del programa, es fundamental la configuración de plugins, es importante tener configurados los plugins de JFlex y CUP en el archivo pom.xml, especificando las rutas a los archivos de definición de la gramática y las reglas léxicas, y las rutas de salida donde se genera automáticamente los analizadores léxicos y sintácticos al compilar el proyecto.

Librerías usadas

Se desarrolla un scanner utilizando la herramienta JFlex utilizando la opción %cup, con el fin que los tokens retornados de tipo Symbol utilizando la clase sym. Por esto, se utilizan las librerías de JFlex y Java CUP que contienen las clases y métodos necesarios para el funcionamiento de los analizadores léxicos y sintácticos.

Cabe mencionar que debido a la estructura utilizada es necesario incluir ciertas dependencias y plugins, a pesar de no ser librerías como tal, para generar automáticamente los analizadores léxicos y sintácticos al compilar el proyecto.

Uno de los plugins utilizados es jflex-maven-plugin, que permite integrar la generación de analizadores léxicos basados en archivos de definición de gramática con el ciclo de vida de construcción de Maven. Asimismo, se establece la ubicación de los archivos de definición de gramática JFlex y se especifica el directorio de salida para el código generado. En este caso, el código Java resultante se coloca en src/main/java.

```

<!-- plugin JFlex -->
<plugin>
  <groupId>de.jflex</groupId>
  <artifactId>jflex-maven-plugin</artifactId>
  <version>1.9.1</version>
  <executions>
    <execution>
      <goals>
        <goal>generate</goal>
      </goals>
      <configuration>
        <outputDirectory>src/main/java</outputDirectory>
        <lexDefinitions>
          <lexDefinition>src/main/jflex</lexDefinition>
        </lexDefinitions>
      </configuration>
    </execution>
  </executions>
</plugin>

```

Otro de los plugins utilizados es el cup-maven-plugin, junto con su dependencia asociada, utilizado para generar analizadores sintácticos basados en CUP. Además, se especifica el directorio de salida para el código generado. En este caso, el código Java resultante se coloca en src/main/java..

```

<!-- plugin Cup -->
<plugin>
  <groupId>com.github.vbmacher</groupId>
  <artifactId>cup-maven-plugin</artifactId>
  <version>11b-20160615-2</version>
  <executions>
    <execution>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <outputDirectory>src/main/java</outputDirectory>
  </configuration>
</plugin>

```

```

<!-- Dependencia Cup -->
<dependencies>
  <dependency>
    <groupId>com.github.vbmacher</groupId>
    <artifactId>java-cup-runtime</artifactId>

```

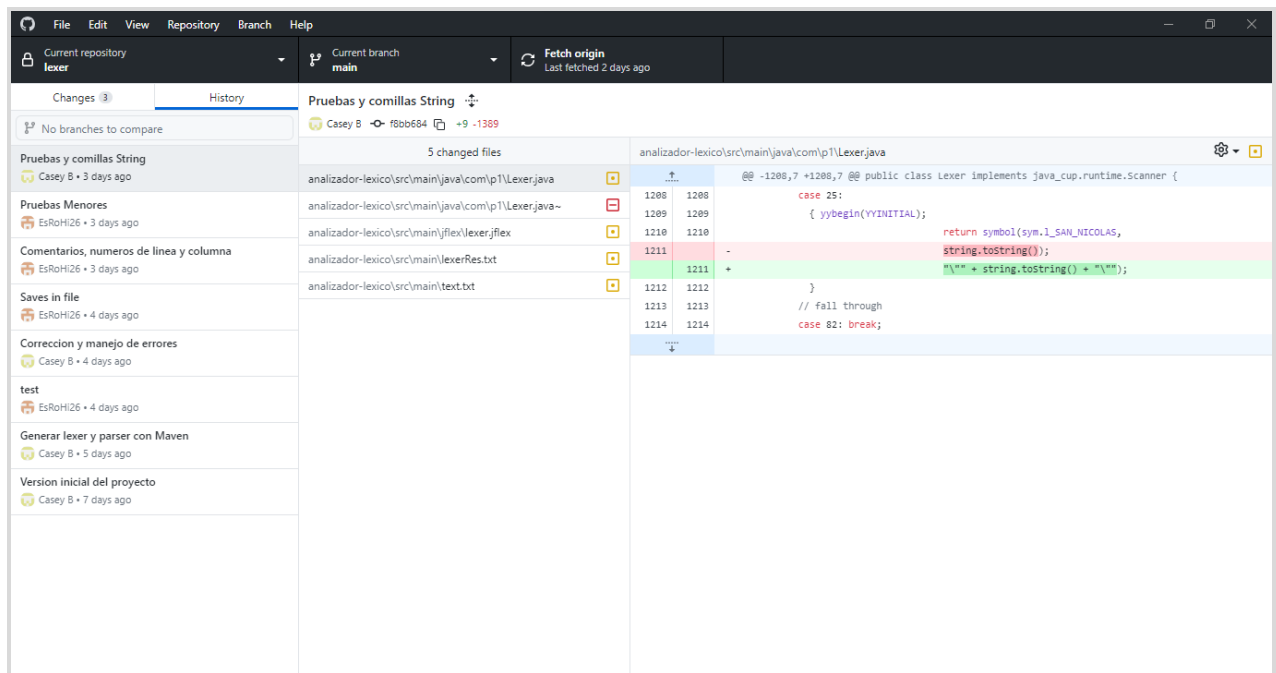

<version>11b-20160615-2</version>
</dependency>

Análisis de resultados

El siguiente checklist muestra los objetivos alcanzados y no alcanzados:

- ☒ Operadores aritméticos binarios
- ☒ Operadores aritméticos unarios
- ☒ Operadores relacionales
- ☒ Operadores lógicos
- ☒ Identificado
- ☒ Tipo
- ☒ Literales
- ☒ Paréntesis
- ☒ Paréntesis cuadrado
- ☒ Llaves
- ☒ Lexemas de estructuras de control (if, elif, else, for, do, until, return, break)
- ☒ Print, read
- ☒ Lexema de fin de expresión
- ☒ Lexema de asigna
- ☒ Lexema separador (coma)

Bitácora



The screenshot shows a Bitácora (Log) window with a dark theme. The top bar includes a menu (File, Edit, View, Repository, Branch, Help) and a status bar showing the current repository (lexer), current branch (main), and a fetch origin button. The main area is divided into two panes. The left pane, titled 'Changes (3)', shows a list of changes with icons for each: 'Pruebas y comillas String' (Casey B, 3 days ago), 'Pruebas Menores' (EsRoHi26, 3 days ago), 'Comentarios, numeros de linea y columna' (EsRoHi26, 3 days ago), 'Saves in file' (EsRoHi26, 4 days ago), 'Correccion y manejo de errores' (Casey B, 4 days ago), 'test' (EsRoHi26, 4 days ago), 'Generar lexer y parser con Maven' (Casey B, 5 days ago), and 'Version inicial del proyecto' (Casey B, 7 days ago). The right pane shows a diff view for the file 'analizador-lexico/src/main/java/com/pt/Lexer.java'. It displays a table with line numbers and the corresponding code changes. The changes include adding a new case (case 25) and modifying the string handling logic.

Line	Old	New
1208		@@ -1208,7 +1208,7 @@ public class Lexer implements java_cup.runtime.Scanner {
1209		case 25:
1210		{ yybegin(VYINITIAL);
1211		return symbol(sym.L_SAN_NICOLAS,
1212		string.toString());
1213		"\" + string.toString() + "\"");
1214		}
1215		// fall through
1216		case 82: break;