Instituto Tecnológico de Costa Rica
Escuela de Computación
IC-5701 Compiladores e Intérpretes GR 60

Documentación Proyecto 1

Profesor:
Allan Rodríguez Dávila

Trabajo elaborado por:
Brasly Villarebia Morales - 2023105915
Sebastián Rodríguez Sánchez - 2023074446

25 de diciembre del 2025

Índice

Manual de usuario

1.1 Requisitos del sistema

- **Sistema operativo:** Windows, Linux o macOS.
- **Lenguaje:** Java JDK 8 o superior.
- **Herramientas:**
    - JFlex
    - Java CUP
    - Git (para la bitácora)

1.2 Estructura del proyecto

PP1_SebastianRodriguez_BraslyVillarebia.zip

- info.txt
- documentacion/
    - Documentacion_Externa.pdf
- Programa/
    - librerias/
        - java-cup-11b-runtime.jar
        - java-cup-11b.jar
        - jflex-full-1.9.1.jar
    - Scripts/
        - Lexer.flex — archivo fuente del analizador léxico (JFlex)
        - Lexer.java — código generado por JFlex
        - Lexer.class — clase compilada del analizador léxico
        - Parser.cup — archivo fuente del analizador sintáctico (CUP)
        - Parser.java — código generado por CUP
        - Parser.class — clase compilada del analizador sintáctico
        - ParserCUPParser$actions.class — clase auxiliar generada por CUP
        - sym.java — definición de símbolos generada por CUP
        - sym.class — clase compilada de símbolos
        - Main.java — clase principal del programa
        - Main.class — clase compilada principal
        - pruebas.txt — archivo de entrada para pruebas
        - tokens.txt — salida de tokens

1.3 Compilación

1. **Generación del Lexer**, el siguiente comando funciona para generar el analizador léxico del lenguaje, a partir del Lexer.Flex en el cual se encuentran las expresiones regulares y las reglas léxicas. Se encargará de leer el archivo fuente, caracter por caracter, identificando los lexemas válidos.
Comando: *java -jar ../Librerias/jflex-full-1.9.1.jar Lexer.flex*
2. **Generación del Parser**, el comando: *java -jar ../Librerias/java-cup-11b.jar -parser Parser Parser.cup*. Ejecuta el Java CUP, permitiendo la generación del Parser.java y el sym.java, los cuales se

encargan de contener las reglas gramaticales y el análisis sintáctico, y de definir los identificadores numéricos de los tokens reconocidos, respectivamente.
3. Para compilar se usa el comando: *javac -cp ".:../Librerias/*" *.java*. El cual debería de generar todos los .class (si compila exitosamente). javac -cp ".;../Librerias/*" *.java (para Windows).
4. Finalmente, para ejecutar se usa el comando: *java -cp ".:../Librerias/*" Main pruebas.txt*. Para poder pasar el archivo *pruebas.txt* como argumento y reporta los tokens reconocidos. (java -cp ".;../Librerias/*" Main pruebas.txt) para Windows. Debido a un "problema" que surgió se ajustó a ISO 8859-1 para un integrante y UTF-8 para otro, sin embargo funcionan de igual manera.

1.4 Flujo de ejecución

Entrada: archivo fuente (pruebas.txt)

Salida: archivo tokens.txt con tokens reconocidos en formato:

Código

Token: <tipo> Lexema: <valor>

Errores léxicos: se imprimen en consola con línea y columna.

Pruebas de funcionalidad

PruebasFuncionalidad P01.mp4

Descripción del problema

Este primer entregable consiste en el desarrollo de un Analizador Léxico para un lenguaje imperativo orientado a la configuración de chips. El objetivo principal es reconocer correctamente los distintos tokens definidos en las especificaciones del proyecto y en la gramática del lenguaje, así como reportar y manejar los errores léxicos encontrados durante el análisis del archivo fuente.

Para la construcción del analizador léxico se utilizan las herramientas JFlex y Java CUP, donde JFlex se encarga de generar el scanner a partir de expresiones regulares, y Java CUP permite la definición formal de los tokens y su integración con el resto del sistema.

Diseño del programa

Lexer.flex

- Define las **expresiones regulares** para reconocer:
  - Palabras reservadas (world, local, Gift, Navidad, ...).
  - Operadores aritméticos, relacionales y lógicos.
  - Delimitadores (¡, !, ¿, ?, endl, ->, ,).
  - Literales (int, float, boolean, char, string).
  - Identificadores válidos.
  - Comentarios de una línea (|) y de bloque (є ... э).
- Incluye directivas %line y %column para reportar posición exacta de cada token.

- Maneja **errores léxicos** con la regla [^], que captura cualquier carácter inválido y lo reporta con línea y columna, aplicando recuperación en modo pánico.

Parser.cup

- Define los **terminales** alineados con los tokens del lexer.
- Establece **precedencias** para operadores aritméticos y unarios (UMINUS).
- Implementa reglas básicas de expresiones (expr_list, expr) para validar integración con el lexer.
- Genera las clases Parser.java y sym.java, que permiten la comunicación entre el scanner y el parser.

Main.java

- Coordina la ejecución del sistema:
    - Lee el archivo fuente (pruebas.txt).
    - Invoca al **Lexer** para obtener tokens.
    - Escribe cada token en tokens.txt con su tipo y lexema.
- Maneja excepciones de entrada/salida y reporta errores de forma clara.
- Finaliza mostrando un mensaje de confirmación: *"Análisis léxico finalizado."*

Algoritmo JFlex

- Convierte las expresiones regulares en un **DFA**.
- El DFA recorre el archivo fuente carácter por carácter y retorna tokens al alcanzar estados de aceptación.
- Se aplica la política de **máxima coincidencia** y prioridad por orden de reglas para resolver ambigüedades (ejemplo: // antes que /).

Recuperación de errores

- La regla [^] captura cualquier carácter no reconocido.
- Se reporta el error con línea y columna.
- El analizador continúa procesando el resto del archivo.

Interacción entre componentes

- **Lexer.flex → Lexer.java:** genera el scanner que tokeniza el código fuente.
- **Parser.cup → Parser.java + sym.java:** define la gramática y los símbolos que recibe el parser.
- **Main.java:** integra ambos, ejecuta el análisis léxico y produce la salida en archivo.

Librerías usadas

- **JFlex 1.9.1**
  Utilizada para la generación del analizador léxico a partir del archivo Lexer.flex.
  Comando de uso: java -jar ../Librerias/jflex-full-1.9.1.jar Lexer.flex
- **Java CUP 11b**
  Utilizada para la generación del parser y de la clase sym, permitiendo la definición formal de los tokens reconocidos por el analizador léxico.
  Comando de uso: java -jar ../Librerias/java-cup-11b.jar -parser Parser Parser.cup

- **Bibliotecas Java CUP Runtime**
  Incluidas dentro del classpath para el uso de la clase java_cup.runtime.Symbol y la interfaz Scanner.
- **Java estándar (JDK 8 o superior)**
  Utilizado para la compilación y ejecución del proyecto, así como para el manejo de archivos y salida estándar.

Para compilar y ejecutar el proyecto incluyendo las librerías externas se utilizan los siguientes comandos: javac -cp ".:../Librerias/*" *.java y java -cp ".:../Librerias/*" Main pruebas.txt

Análisis de resultados

Se cumplió con la totalidad del proyecto, incluyendo todos los puntos solicitados, entre los cuales se consideran:

- Reconocer todos los tokens definidos en la gramática (operadores, delimitadores, palabras reservadas, literales e identificadores).
- Generar correctamente el archivo *tokens.txt* con tipo y lexema de cada token.
- Implementar el manejo de errores léxicos con recuperación en modo pánico (línea y columna).
- Integrar el lexer con CUP, incluyendo precedencias y reglas básicas de expresiones.
- El programa principal (Main.java) lee archivos fuente, invoca el lexer y produce la salida.
- Realizar pruebas completas con ejemplos de todos los tipos de tokens y errores.
- Entregar documentación interna y externa conforme a lo solicitado.
- Usar GitHub para control de versiones y bitácora de commits.
- Cumplir con la estructura y requisitos administrativos de la entrega.

Bitácora

sebro08/CompiladoresJFlex: Proyecto de compiladores e intérpretes en verano, en JFlex y Cup