



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

**Fase 1**

Isaac Ramírez Rojas

Adrian José Villalobos Peraza

Compiladores e Intérpretes

Verano 2024

<b>Manual de Usuario.....</b>	<b>3</b>
Instrucciones de compilación.....	3
Ejecución.....	4
Uso.....	4
<b>Pruebas de Funcionalidad.....</b>	<b>5</b>
<b>Descripción de Problema.....</b>	<b>7</b>
<b>Diseño del Programa.....</b>	<b>7</b>
Decisiones de Diseño.....	8
Algoritmos Usados.....	8
<b>Librerías Usadas.....</b>	<b>8</b>
Librerías de manejo de archivos.....	8
JavaCup.....	8
JFlex.....	9
<b>Análisis de Resultados.....</b>	<b>9</b>
<b>Bitácora.....</b>	<b>11</b>

# Manual de Usuario

## Instrucciones de compilación

Para llevar a cabo la compilación del proyecto es indispensable contar con las siguientes herramientas y librerías:

- **Java (JDK):** Asegúrese de tener instalado el Java Development Kit en una versión adecuada (por ejemplo, la versión 8 o superior). Las pruebas realizadas a este proyecto fueron con Oracle OpenJDK 23.0.1
- **Gradle:** Esta herramienta de automatización de construcción facilitará la resolución de dependencias, la ejecución de tareas de compilación y la empaquetación del proyecto. Adicionalmente, se puede utilizar en la línea de comandos para la ejecución del proyecto.
- **JavaCup:** Esta librería se utiliza para generar el analizador sintáctico (parser). A partir de la gramática definida, JavaCup se encarga de producir el código Java que valida la estructura sintáctica del programa fuente. En este caso es requerida únicamente para proporcionar los símbolos requeridos por el lexer, por medio del archivo `sym.java`.
- **JFlex:** JFlex permite generar el analizador léxico (lexer), el cual convierte el flujo de caracteres del código fuente en tokens. Se utiliza para producir el `Lexer.java`.

Antes de iniciar la compilación, es importante revisar la estructura de archivos del proyecto. Asegúrese de que el archivo `Lexer.java`, así como `parser.java` y `sym.java` (generados a partir de las especificaciones de JFlex y JavaCup), se encuentren en la carpeta `parser`. Esto es estrictamente necesario

En cuanto al entorno de desarrollo, las pruebas realizadas se llevaron a cabo utilizando IntelliJ IDEA de JetBrains. Esta herramienta integrada permite compilar, ejecutar y depurar el proyecto de forma intuitiva. Si ha optado por utilizar IntelliJ, puede ejecutar el proceso de compilación directamente desde su interfaz, de lo contrario puede ser por línea de comandos con gradle. Cabe destacar que el proyecto fue ejecutado y probado en entornos Linux y MacOS, se desconoce su funcionamiento en sistemas operativos Windows.

## Ejecución

La ejecución del proyecto puede llevarse a cabo desde el entorno de desarrollo integrado IntelliJ IDEA. Una vez que se hayan satisfecho todas las dependencias, completado el proceso de compilación y ubicado los archivos generados en las carpetas correspondientes, es posible ejecutar el programa con el botón dedicado a esta acción.

Para ello, abra el proyecto en IntelliJ IDEA y asegúrese de que el archivo `Main.java` esté presente y listo para su ejecución. Puede verificar la configuración de la clase principal en las opciones de ejecución del IDE. Finalmente, presione el botón de “Run” (Ejecutar) ubicado en la parte superior de la interfaz o utilice el atajo de teclado correspondiente (por ejemplo, `Shift + F10` en Windows). IntelliJ IDEA se encargará de iniciar la aplicación, mostrando la salida y, en caso de haber un analizador léxico y sintáctico definido, procesa el código fuente ingresado, reportando los tokens generados, los posibles errores o el resultado final de la ejecución.


## Uso

El programa está diseñado para procesar archivos de texto y realizar un análisis léxico sobre su contenido, proporcionando información estructurada acerca de los elementos del lenguaje presentes en dicho archivo. Para usar el programa, primero el usuario debe ingresar algunas de las opciones disponibles (del 1 al 3, generar archivos necesarios, probar el lexer o salir). En caso de seleccionar la segunda opción el usuario únicamente debe proporcionar un archivo de texto válido, cuya ruta deberá ser especificada como entrada en el momento de la ejecución. La ruta proporcionada debe ser fiel al root del proyecto.

Una vez proporcionado el archivo, el programa procederá a abrirlo y leer su contenido línea por línea. A medida que procesa el texto, realiza un análisis léxico, identificando y clasificando los componentes del lenguaje según las reglas definidas en la gramática. Esto incluye el reconocimiento de palabras clave, identificadores, operadores, literales y otros elementos significativos del lenguaje. Todo esto se imprime en consola pero, también se genera un archivo `.txt` llamado salida, en la carpeta `lexer`, donde se puede apreciar todo lo anterior (importante destacar que los errores solo se muestran en consola y no en el archivo de output). El archivo de output se encontrará en el directorio `src/lex` bajo el nombre de `salida.txt`.

# Pruebas de Funcionalidad

En el video a continuación se detallan todas las pruebas realizadas para evaluar la funcionalidad del programa. Estas pruebas incluyen ejemplos prácticos de análisis léxico y sintáctico, casos con entradas válidas e inválidas, y la verificación del comportamiento esperado del programa en diferentes escenarios. El video muestra tanto la ejecución como los resultados obtenidos, brindando una visión completa del correcto funcionamiento del sistema.

 Pruebas-Funcionales.mov

Importante: al momento de grabar el video, aun no estaba implementado el menú, por lo que se proporcionan screenshots adicionales.

```
> Task :compiler.Main.main()
1. Generar lexer y otros archivos necesarios
2. Usar el lexer
3. Salir
Seleccione una opcion: 2
Ingrese la ruta del archivo de prueba (debe ser un archivo .txt): src/lex/test03.txt
```

```

Linea: 26 Columna: 8 Token: OPEN_PAREN Valor: abreregalo
Linea: 26 Columna: 19 Token: INTEGER Valor: rodolfo
Linea: 26 Columna: 27 Token: IDENTIFIER Valor: _i_
Linea: 26 Columna: 31 Token: EQ Valor: entrega
Linea: 26 Columna: 39 Token: INTEGER_LITERAL Valor: 0
Linea: 26 Columna: 41 Token: SEMICOLON Valor: finregalo
Linea: 26 Columna: 51 Token: IDENTIFIER Valor: _i_
Linea: 26 Columna: 55 Token: GT Valor: minstix
Linea: 26 Columna: 63 Token: INTEGER_LITERAL Valor: 3
Linea: 26 Columna: 65 Token: SEMICOLON Valor: finregalo
Linea: 26 Columna: 77 Token: PLUS Valor: navidad
Linea: 26 Columna: 85 Token: INTEGER_LITERAL Valor: 1
Linea: 26 Columna: 87 Token: CLOSE_PAREN Valor: cierraregalo
Linea: 26 Columna: 100 Token: OPEN_BLOCK Valor: abrecuento
Linea: 27 Columna: 1 Token: PRINT Valor: narra
Linea: 27 Columna: 18 Token: STRING_LITERAL Valor: Iteracion:
Linea: 27 Columna: 20 Token: SEMICOLON Valor: finregalo
Linea: 28 Columna: 1 Token: CLOSE_BLOCK Valor: cierracuento
Linea: 30 Columna: 1 Token: CLOSE_PAREN Valor: cierraregalo

```

BUILD SUCCESSFUL in 9s

2 actionable tasks: 1 executed, 1 up-to-date

Esto está mal: <contador> en línea: 9, columna: 9

Esto está mal: <.> en línea: 10, columna: 30

Esto está mal: <tru> en línea: 11, columna: 23

Esto está mal: <\_texto> en línea: 12, columna: 8

Esto está mal: <fals> en línea: 23, columna: 16

Esto está mal: <i> en línea: 26, columna: 75

```

Main.java  salida.txt  minijava.jflex  Tester.java  Lexer.java  test01.txt
66 Linea: 22 Columna: 1 Token: PRINT Valor: narra
67 Linea: 22 Columna: 24 Token: STRING_LITERAL Valor: Dentro del while
68 Linea: 22 Columna: 26 Token: SEMICOLON Valor: finregalo
69 Linea: 23 Columna: 1 Token: IDENTIFIER Valor: _flag_
70 Linea: 23 Columna: 8 Token: EQ Valor: entrega
71 Linea: 23 Columna: 21 Token: SEMICOLON Valor: finregalo
72 Linea: 24 Columna: 1 Token: CLOSE_BLOCK Valor: cierracuento
73 Linea: 26 Columna: 1 Token: FOR Valor: duende
74 Linea: 26 Columna: 8 Token: OPEN_PAREN Valor: abreregalo
75 Linea: 26 Columna: 19 Token: INTEGER Valor: rodolfo
76 Linea: 26 Columna: 27 Token: IDENTIFIER Valor: _i_
77 Linea: 26 Columna: 31 Token: EQ Valor: entrega
78 Linea: 26 Columna: 39 Token: INTEGER_LITERAL Valor: 0
79 Linea: 26 Columna: 41 Token: SEMICOLON Valor: finregalo
80 Linea: 26 Columna: 51 Token: IDENTIFIER Valor: _i_
81 Linea: 26 Columna: 55 Token: GT Valor: minstix
82 Linea: 26 Columna: 63 Token: INTEGER_LITERAL Valor: 3
83 Linea: 26 Columna: 65 Token: SEMICOLON Valor: finregalo
84 Linea: 26 Columna: 77 Token: PLUS Valor: navidad
85 Linea: 26 Columna: 85 Token: INTEGER_LITERAL Valor: 1
86 Linea: 26 Columna: 87 Token: CLOSE_PAREN Valor: cierraregalo
87 Linea: 26 Columna: 100 Token: OPEN_BLOCK Valor: abrecuento
88 Linea: 27 Columna: 1 Token: PRINT Valor: narra
89 Linea: 27 Columna: 18 Token: STRING_LITERAL Valor: Iteracion:
90 Linea: 27 Columna: 20 Token: SEMICOLON Valor: finregalo
91 Linea: 28 Columna: 1 Token: CLOSE_BLOCK Valor: cierracuento
92 Linea: 30 Columna: 1 Token: CLOSE_PAREN Valor: cierraregalo
93

```

# Descripción de Problema

El problema consiste en crear un analizador léxico para la gramática solicitada. Para esto, se debe desarrollar un scanner utilizando distintas herramientas y librerías, específicamente, el lenguaje de programación java y las librerías jflex y cup.

El analizador léxico generado debe ser capaz de leer un archivo fuente, destacar todos los tokens encontrados y su identificador asociado con el lexema. Asimismo debe manejar los errores de forma exitosa, por lo que se debe implementar una técnica de recuperación en modo pánico, la cual consiste en detectar el error, notificarlo (lo cual incluye al error en sí, la línea y columna en la que se encuentra) y seguir analizando el resto del archivo.

Para la creación del lexer (scanner), es necesario analizar la especificación de la gramática del lenguaje, de modo que se identifiquen sus terminales, literales e identificadores. Para globalizar estos términos, es necesario utilizar definiciones regulares.

Por otro lado, se solicita la creación de un “menú”, de modo que se proporcionen las distintas opciones del programa (como generar los archivos o probar estos). Este menú será implementado en terminal, sin interfaz gráfica. En caso de seleccionar la opción de probar, se debe proporcionar una ruta válida a un archivo .txt (si es inválida, el programa lo hará saber) y los resultados del análisis léxico serán mostrados tanto en consola como en un archivo de output con terminación .txt.

En adición a todo lo anterior, se solicita que el proyecto y la solución del problema sea abordado usando un controlador de versiones (en este caso GitHub) para llevar un control total de los cambios realizados al proyecto.

# Diseño del Programa

El diseño del presente programa se enfoca en desarrollar la fase de Análisis Léxico para un lenguaje imperativo ligero, destinado a la configuración de sistemas empujados. La implementación fue cuidadosamente estructurada siguiendo decisiones de diseño orientadas a garantizar eficiencia, modularidad y facilidad de mantenimiento.

Estas decisiones se ven reflejadas en la selección de herramientas como JFlex y CUP, que permiten generar automáticamente el scanner y manejar los tokens. Así como una modularización correcta del programa, para separar las distintas funcionalidades de este.

## Decisiones de Diseño

Uno de los principales desafíos enfrentados durante el desarrollo fue el manejo de errores léxicos, ya que era necesario reportarlos sin detener la ejecución del compilador. Para resolver esto, se implementó una técnica de fallback en JFlex, donde se configuró una sección especial que captura cualquier símbolo no reconocido y lo reporta indicando la línea y columna de aparición, permitiendo así la recuperación en modo pánico y la continuidad del análisis. Además, se desarrolló una función que almacena los tokens recolectados en un archivo `.txt`, con el fin de contar con un registro detallado de los tokens identificados, incluyendo su tipo y lexema. Estas decisiones garantizaron un manejo robusto de errores y facilitaron la verificación y depuración del funcionamiento del scanner.

## Algoritmos Usados

No se utilizó ningún algoritmo en sí, lo que se utilizó para este proyecto fue una base proporcionada por las documentaciones de jflex y cup. Sobre la base anteriormente mencionada se empezó a formar el lexer y el resto del proyecto.

## Librerías Usadas

### Librerías de manejo de archivos

Estas librerías se utilizan para la gestión y organización de los archivos generados durante el proceso de construcción del compilador, en particular aquellos producidos por JFlex y JavaCup. Por ejemplo, una vez que JFlex y JavaCup han generado los archivos `Lexer.java`, `parser.java` y `sym.java`, se emplean funcionalidades de `java.nio.file` y manejo de excepciones `IOException` para mover, reubicar o sobrescribir estos ficheros en las rutas correspondientes del proyecto, asegurando así una correcta estructura interna del compilador.

## JavaCup

JavaCup es una herramienta que se utiliza para generar el analizador sintáctico, la siguiente etapa después del análisis léxico. Una vez que JFlex ha proporcionado la secuencia de tokens, el parser generado por JavaCup toma estos tokens y los combina de acuerdo con las reglas gramaticales definidas en el archivo de especificaciones (`.cup`). Este archivo describe la gramática en notación BNF o similar, detallando cómo los tokens deben organizarse para conformar sentencias y estructuras sintácticas válidas del lenguaje.



El objetivo principal de JavaCup es verificar si la secuencia de tokens cumple con las reglas de la gramática, sin embargo, como se mencionó anteriormente, en este proyecto su uso se reduce al proporcionamiento de los símbolos requeridos.

## JFlex

JFlex es una herramienta que automatiza la creación del analizador léxico, es decir, se encarga de traducir el texto fuente del programa en una secuencia de tokens. Un token representa la unidad mínima de significado en un lenguaje de programación (por ejemplo, palabras clave, identificadores, literales, símbolos de operación, etc.). JFlex toma como entrada un archivo de especificaciones (normalmente con extensión `.jflex`) donde se definen las expresiones regulares que describen cada tipo de token. A partir de estas especificaciones, JFlex genera una clase Java (`Lexer.java`) que, al ejecutarse, lee el código fuente carácter por carácter y crea la lista ordenada de tokens para su posterior análisis.

## Análisis de Resultados

Objetivo	Estado	Comentario
Utilizar las herramientas Jflex y Cup para la creación de un lexer que usa la directiva Cup.	Completado	
Agregar las palabras reservadas de la gramática según lo solicitado.	Completado	Se incluyeron todos los lexemas y palabras reservadas solicitadas, además, se agregaron los terminales necesarios.
Leer un archivo de prueba y de este obtener: todos los lexemas, su línea y columna	Completado	Se muestra tanto en terminal como en un archivo .txt.

de aparición, token y valor.		
Leer un archivo de prueba y de este obtener: todos los errores y mostrar su línea y columna de aparición, así como el error en sí.	Completado	Se muestra únicamente en terminal.
Utilizar el modo recuperación en pánico para el manejo de errores.	Completado	El programa es capaz de detectar los errores y mostrarlos, sin detener su funcionamiento.
Crear un menú que funcione como interfaz de usuario. Este cuenta con la opción de generar archivos lexer sym y parser o realizar una prueba.	Completado	Se implementó correctamente el menú. Cabe destacar que las rutas a ingresar son desde el root del proyecto, por ejemplo "src/lex/test03.txt".
Generar un archivo de salida que contiene todos los lexemas encontrados, su línea y columna de aparición, valor y token.	Completado	Este archivo es generado en el directorio "src/lex" bajo el nombre "salida.txt".
Utilizar GitHub como gestor de versiones.	Completado	<a href="https://github.com/akhodz/Christmas-Compiler">https://github.com/akhodz/Christmas-Compiler</a>
Documentar internamente el	Completado	Se utilizó la directiva

código, indicando que recibe, hace y devuelve una función o método.		Javadoc para la documentación de este proyecto.
---	--	---

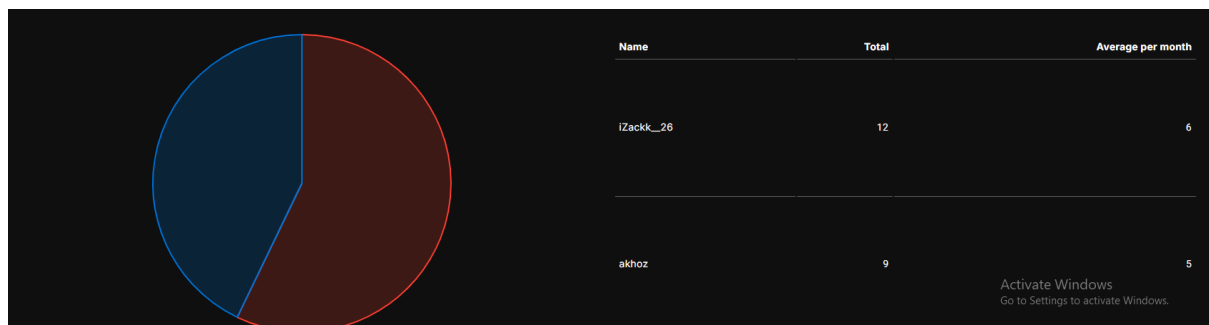
## Bitácora

En esta sección se encuentran todos los commits realizados en el repositorio, presentados de manera estructurada y descriptiva para su fácil consulta. Para una visualización más detallada y formal, puede utilizar la herramienta [GitStats](#).

Al acceder a la página de GitStats, cargue la información del repositorio utilizando el archivo JSON proporcionado en el siguiente enlace: [Archivo JSON del repositorio](#). Esto le permitirá explorar los datos del repositorio de manera interactiva, incluyendo detalles como los commits, autores, mensajes y fechas asociadas.

Los estudiantes son:

- iZack\_26: Isaac Ramirez
- akhoz: Adrián Villalobos



Project	Time	Title	Author	+	-		
Christmas-Compiler	Dec 16, 2024, 9:56...	feat: Added logs on salida.txt file	IZackk_26	384	107		
Christmas-Compiler	Dec 15, 2024, 3:37...	feat: adding tokens names, main menu and info.txt	akhoz	137	71		
Christmas-Compiler	Dec 14, 2024, 9:51...	feat: adding javadoc and new test file	akhoz	326	128		
Christmas-Compiler	Dec 13, 2024, 2:10...	feat: Added new test files	IZackk_26	281	82		
Christmas-Compiler	Dec 13, 2024, 1:57...	feat: Added new error exception to the .jflex file	IZackk_26	1753	711		
Christmas-Compiler	Dec 13, 2024, 1:29...	feat: Added organizer module and organize class	IZackk_26	1648	1184		
Christmas-Compiler	Dec 13, 2024, 11:17...	feat: added booleand_literal	IZackk_26	1775	1474		
Christmas-Compiler	Dec 13, 2024, 11:11...	feat: Added new rules for .jflex files and fixed some errors	IZackk_26	3899	1102		
Christmas-Compiler	Dec 13, 2024, 9:4...	feat: adding error handling and line/column indicators	akhoz	1365	1336		
Christmas-Compiler	Dec 11, 2024, 7:25...	feat: adding public features to Lexer class	akhoz	1418	1158		
Christmas-Compiler	Dec 11, 2024, 7:05...	feat: fixing lexer error, now it works, still need to check the grammar	akhoz	1568	2422		
Christmas-Compiler	Dec 11, 2024, 4:40...	fix: still need to fix Lexer errors	akhoz	2092	442		
Christmas-Compiler	Dec 11, 2024, 3:20...	feat: Added new version of project (functional)	IZackk_26	1521	23		
Christmas-Compiler	Dec 11, 2024, 2:48...	fix: new files	IZackk_26	44	152		
Christmas-Compiler	Dec 11, 2024, 2:12 ...	feat: New Structure	IZackk_26	54	2228		
Christmas-Compiler	Dec 11, 2024, 1:55 ...	feat: added commands to run the lexer and sym	IZackk_26	1276	62		
Christmas-Compiler	Dec 11, 2024, 1:47 ...	feat: adding jflex initial set up	akhoz	1210	0		
Christmas-Compiler	Dec 11, 2024, 1:27 ...	feat: Added jflex file	IZackk_26	87	0		
Christmas-Compiler	Dec 11, 2024, 1:20 ...	feat: adding library files	akhoz	0	0		
Christmas-Compiler	Dec 11, 2024, 1:12 ...	feat: Added PDF document	IZackk_26	0	0		
				Previous	1	2	Next

URL del repositorio: [GitHub](#)