



## Primer Proyecto (Scanner con JFlex)

### **Profesora:**

Ericka Marín Schumann

### **Integrantes:**

Allison Montero Merlo	2019044431
Gabriela Gutiérrez Valverde	2019024089
Francisco Chavarro	2019227569

**Compiladores e Intérpretes (IC- 5701)**

**Escuela de Ingeniería en Computación**

**Grupo 3**

**27 de Septiembre de 2022**

# Índice

<b>Introducción</b>	<b>3</b>
<b>Estrategia de solución</b>	<b>4</b>
<b>Análisis de resultados</b>	<b>6</b>
<b>Lecciones aprendidas</b>	<b>7</b>
Francisco Chavarro Conde	7
Allison Montero Merlo	7
Gabriela Gutiérrez	8
<b>Casos de prueba</b>	<b>9</b>
Caso de prueba #1	9
Caso de prueba #2	9
Caso de prueba #3	9
<b>Manual de usuario</b>	<b>10</b>
<b>Bitácora de trabajo</b>	<b>11</b>
<b>Bibliografía</b>	<b>12</b>
<b>Evidencias</b>	<b>12</b>

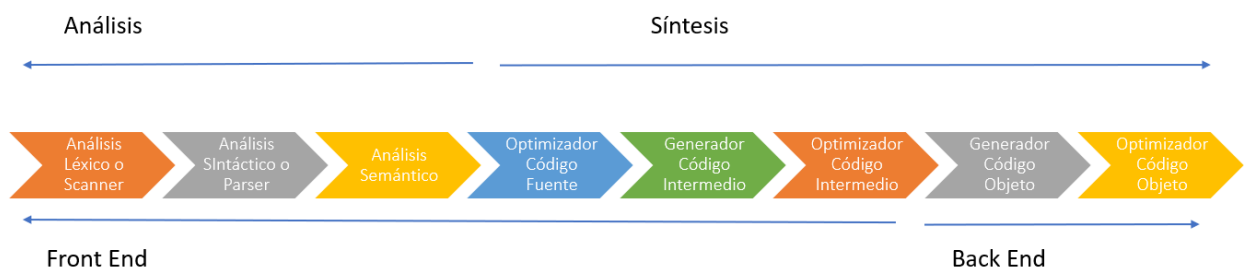
# Introducción

Muchas aplicaciones por no decir casi que todas, necesitan validar los datos de entrada, sobre todo cuando lo que se espera del usuario es texto o algo similar al texto, por ello, es importante conocer cómo sucede el proceso de verificar estas entradas, ya que para el correcto funcionamiento es necesario que los datos introducidos sean precisos y correctos. Bajo este mismo concepto es que funcionan los compiladores, en donde hay varias etapas para su funcionamiento, recordemos que un compilador es un programa o básicamente un traductor que nos permite pasar de un lenguaje fuente a un lenguaje objeto, y se divide en varias etapas para realizar este objetivo, la primera de estas es el análisis léxico, que es la etapa donde se verifica que todos los caracteres o el texto en general sea permitido por el programa, precisamente esta es la etapa que se espera desarrollar en este primer proyecto, en como idea del curso aprenderemos cómo funciona y cómo desarrollar y programar un analizador léxico. Así pues en este proyecto se espera realizar un analizador léxico en su totalidad, siguiendo las indicaciones específicas que nos dieron y desarrollado en java, para este caso en específico el analizador va a ser capaz de recibir un programa en lenguaje C.

# Estrategia de solución

Lo primero que se debe de tener en cuenta es tener en claro la manera en la que funciona un compilador y todas sus partes. Ver figura 1.

Figura 1 - Partes de un compilador



Fuente: Ericka Marin Schumman

Una vez sabiendo eso podemos continuar, pero bueno, inicialmente para este primer proyecto es necesario realizar el análisis léxico, que es la primera etapa del compilador como pudimos observar en la figura anterior, en esta etapa lo que se debe de hacer es analizar que todos los caracteres o todo el texto en general sea permitido para el lenguaje, como es la primera etapa debe hacerse con sumo cuidado ya que el resultado de este análisis será con lo que trabajarán el resto de etapas.

Ahora bien, el primer paso para realizar el analizador léxico y que debíamos de seguir es tener en claro que lenguaje íbamos a permitir, es decir, el lenguaje fuente, en este caso era el lenguaje de "C". Una vez tenemos claro el lenguaje fuente podemos pasar a definir lo que será nuestro lexer, en donde debemos indicar todas las estructuras permitidas, así como lo "tokens", este último término lo usaremos para definir cualquier elemento del lenguaje fuente. De esta manera procedemos a realizar categorías según sea, para este proyecto contábamos con una indicación, por lo cual dividimos los tokens en los siguientes grupos: identificadores, literales, palabras reservadas y operadores. Mínimo eran estos, sin embargo se puede ser más específico si así se desea. Teniendo ya estos grupos hechos hay que definir qué son exactamente, para ello creamos un archivo en donde definimos todos los autómatas de cada uno de los grupos, en este caso lo que definimos fueron las expresiones regulares de cada uno, y con ayuda de la

herramienta “JFlex” se genera un nuevo documento el cual interpreta estas expresiones y las convierte en autómatas, y básicamente así se crea el analizador, una vez las reglas están definidas podemos introducir cualquier token y debería de poder identificarlo como uno de los grupos o bien como un error léxico. Esto para un token individual, luego para poder realizar el análisis a todo el documento solo necesitábamos tomar cada línea como un conjunto de tokens y así poder analizar línea por línea hasta que no haya tokens. Para poder manejar de forma más sencilla los tokens se decidió crear una clase llamada token

Es importante mencionar que para el desarrollo de todo el proyecto y la estrategia seguida se estudió de cerca un tutorial en youtube acerca de cómo funciona la herramienta JFlex, dicho tutorial fue de gran ayuda para poder tener los procedimientos a seguir claros.[1]

Por último con respecto a la interfaz se trató que fuera lo más sencilla posible pero también lo más práctica, y por conocimiento previo se decidió usar java swing, además, que era mucho más fácil para un usuario correrlo sin instalar herramientas externas.

# Análisis de resultados

Primeramente cabe destacar que el proyecto fue finalizado en su totalidad y de una manera exitosa, es decir, se completaron todas las tareas esperadas de una manera exitosa, a continuación, se presenta una lista de las funcionalidades y su porcentaje de realización.

- El programa cuenta con una interfaz gráfica para que el usuario pueda interactuar. Porcentaje: 100%
- El usuario es capaz de seleccionar un archivo desde su equipo para analizarlo. Porcentaje: 100
- El programa tiene todos los grupos necesarios de tokens para poder realizar el análisis. Porcentaje 100%
- El programa es capaz de reconocer las palabras reservadas del lenguaje . Porcentaje 100%
- El programa reconoce todos los operadores necesarios que posee el lenguaje C. Porcentaje 100%
- El programa es capaz de identificar cada uno de los distintos tipos de tokens (identificadores, operadores, literarios, palabras reservadas). 100%
- El programa es capaz de detectar en qué línea se encuentra, ya sea para los tokens o para detectar errores. Porcentaje 100%
- El programa es capaz de indicar en qué línea aparece cada token y la cantidad de veces por línea. Porcentaje 100%
- El programa es capaz de reconocer errores léxicos. Porcentaje 100%
- En caso de encontrar un error el programa debe ser capaz de continuar hasta el final del análisis. Porcentaje 100%
- El programa es capaz de encontrar los errores léxicos además de ser capaz de señalar en qué línea ocurrió.

# Lecciones aprendidas

## Francisco Chavarro Conde

### **Hardskills**

- El uso de java swing, a pesar de ser una de las maneras más sencillas y comunes de hacer interfaces en java siempre había usado otras herramientas, es la primera vez que tengo la oportunidad de usar esta herramienta y fue un aprendizaje muy grande con ella.
- La estructura de los compiladores y saber cómo funciona por dentro uno. Al menos la primera parte y con herramientas de ayuda como JFlex.
- El uso de JFlex, para la generación de los autómatas a base de las expresiones regulares hechas anteriormente.

### **Softskills**

- La organización, en este trabajo la organización fue clave para terminarla a tiempo, y en este caso fue así.
- La importancia de la comunicación desde el inicio del proyecto, lo cual ayuda a que todo arranque antes y de una mejor manera.

## Allison Montero Merlo

### **Hardskills**

- Uso de la herramienta JFlex para la generación de autómatas a partir de expresiones regulares.
- Dominio de la creación de expresiones regulares para reconocer un lenguaje de programación.
- Comprensión de las etapas del proceso de compilación más a detalle, en especial la etapa que corresponde al análisis léxico.

### **Softskills**

- La comunicación fue especialmente importante para la etapa de elaboración de las expresiones regulares, además del trabajo en conjunto.
- La organización para repartir las tareas de manera correcta y poder avanzar de forma paulatina.

# Gabriela Gutiérrez Valverde

## **Hardskills**

- Repaso de las expresiones regulares y saber cómo expresarlas en código.
- La herramienta de JFlex para ya hacer la definición del lenguaje y cómo se ejecuta.
- La herramienta que tiene netbeans para desarrollar interfaces de usuario y el cómo se puede tener más de una vista en una pantalla, además de agregar documentos al programa desde el explorador de archivos.

## **Softskills**

- Fue muy importante la comunicación asertiva para conocer la manera en la que trabajamos en el proyecto y saber que todos nos estábamos entendiendo.
- También conocer las virtudes de cada compañero para saber qué tarea le resultaba más sencillo a cada persona de forma que todos colaboramos para el desarrollo del proyecto.



# Casos de prueba

## Caso de prueba #1

**Descripción:** Se desea comprobar que el programa reconoce los saltos de línea y se hace un print "Error" para identificarlos.

**Resultado esperado:** Un mensaje de error por cada salto de línea.

**Resultado obtenido:** Los saltos de línea se identifican correctamente, el output se puede ver en la sección de evidencias. (Ver evidencia 1.1)

## Caso de prueba #2

**Descripción:** Se deseaba comprobar el funcionamiento del programa ya directamente en la interfaz, para ello, se usó un programa muy sencillo (un hola mundo), se había probado previamente en la consola, sin embargo, se esperaba que las tablas se llenaran de una manera correcta.

**Resultado esperado:** Dos tablas con varias columnas, en donde podríamos ver el tipo de token que era, en que línea se encontraba y si había errores, cuál era y en qué línea.

**Resultado obtenido:** Funcionó bastante bien ya que las tablas se llenaron correctamente, además que se identificaron los tokens, sin embargo, hubo un error que era necesario arreglar, todos los tokens decían que estaban en la línea cero, tanto los errores como la parte de los tokens. Ver evidencias.

## Caso de prueba #3

**Descripción:** Se hace la prueba con un programa de ejemplo que contiene caracteres no permitidos por el lenguaje.

**Resultado esperado:** Se espera que el compilador identifique los caracteres inválidos y los clasifique como errores.

**Resultado obtenido:** El Scanner identifica correctamente los errores léxicos encontrados y los clasifica como tal. (Ver evidencia 1.2)

# Manual de usuario

Enlace al manual:  Manual de Usuario.pdf

# Bitácora de trabajo

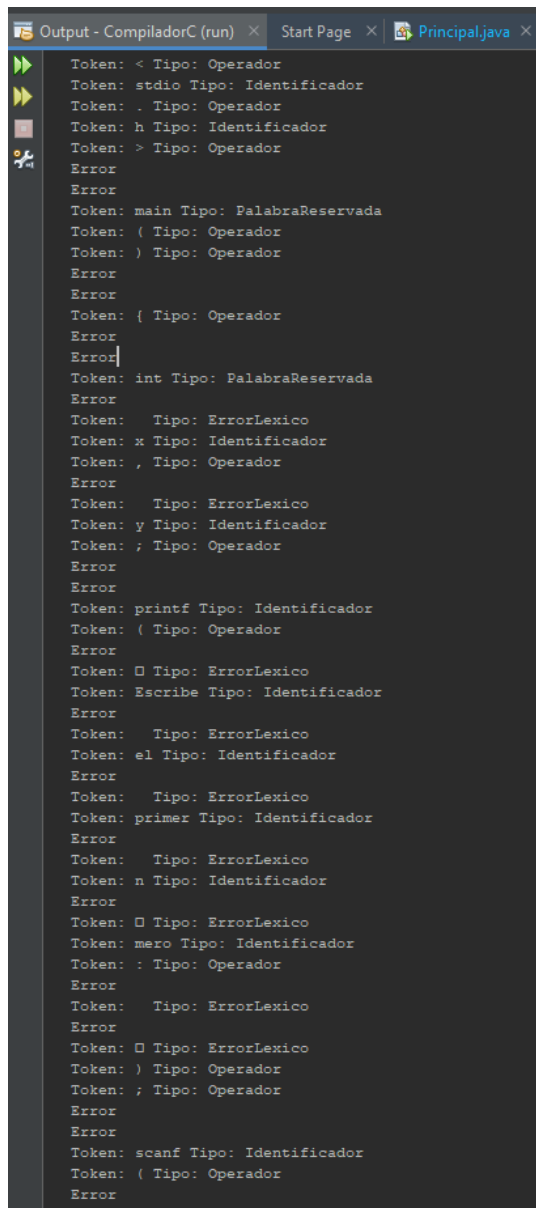
Tarea	Fecha	Participantes
Primera reunión, aquí se habló del proyecto y cómo se abordará	12/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Reunión para la división del trabajo	13/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Creación de los tipos de tokens, además de la estructura del analizador léxico	15/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Creación del proyecto y de las expresiones regulares	16/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Elaboración del primer Lexer y pruebas con el mismo	18/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Elaboración de la clase "modelo" la cual usaba el lexer realizar el análisis. Aquí se lleva el conteo de las líneas y los errores.	19/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Elaboración de la interfaz y de la clase controlador.	21/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Inicio del desarrollo de la documentación	22/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Reunión para ver el estado del proyecto	22/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Implementación de la función de seleccionar archivo	23/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Corrección de errores, por ejemplo el botón de regresar de la tabla no servía	24/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez
Finalización de la documentación	25/09/2022	Francisco Chavarro Allison Montero Gabriela Gutierrez

# Bibliografía

1. [Charlead]. (2019, 17 febrero). *JFlex | Analizador léxico con Java (explicación paso a paso)*. YouTube. Recuperado 24 de septiembre de 2022, de <https://www.youtube.com/watch?v=5mIRn2yEe8>
2. Klein, G. (s. f.). *JFlex - JFlex The Fast Scanner Generator for Java*. Recuperado 27 de septiembre de 2022, de <https://www.jflex.de/index.html>
3. *Using flex in c and regular expressions*. (2015, 6 febrero). Stack Overflow. Recuperado 27 de septiembre de 2022, de <https://stackoverflow.com/questions/28368439/using-flex-in-c-and-regular-expressions>
4. *Literales de un solo carácter*. (2014, 9 septiembre). Linux, C/C++, Apuntes, etc. . . Recuperado 27 de septiembre de 2022, de <https://baulderasec.wordpress.com/programando-2/cc/elementos-del-lenguaje-c/literales/literales-de-un-solo-caracter/>

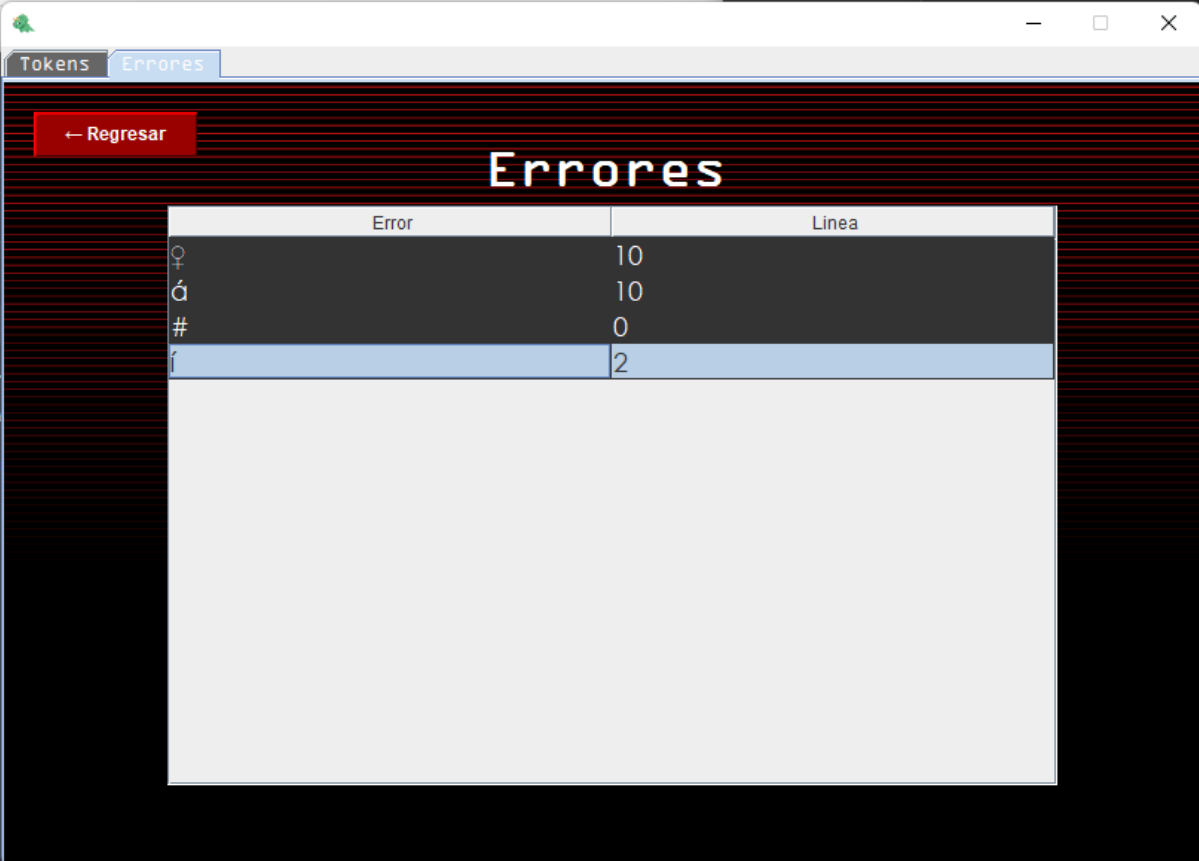
# Evidencias

## 1.1 Evidencia de caso de prueba # 1



```
Output - CompiladorC (run) x Start Page x Principal.java x
Token: < Tipo: Operador
Token: stdio Tipo: Identificador
Token: . Tipo: Operador
Token: h Tipo: Identificador
Token: > Tipo: Operador
Error
Error
Token: main Tipo: PalabraReservada
Token: ( Tipo: Operador
Token: ) Tipo: Operador
Error
Error
Token: { Tipo: Operador
Error
Error
Token: int Tipo: PalabraReservada
Error
Token: Tipo: ErrorLexico
Token: x Tipo: Identificador
Token: , Tipo: Operador
Error
Token: Tipo: ErrorLexico
Token: y Tipo: Identificador
Token: ; Tipo: Operador
Error
Error
Token: printf Tipo: Identificador
Token: ( Tipo: Operador
Error
Token: Tipo: ErrorLexico
Token: Escribe Tipo: Identificador
Error
Token: Tipo: ErrorLexico
Token: el Tipo: Identificador
Error
Token: Tipo: ErrorLexico
Token: primer Tipo: Identificador
Error
Token: Tipo: ErrorLexico
Token: n Tipo: Identificador
Error
Token: Tipo: ErrorLexico
Token: mero Tipo: Identificador
Token: : Tipo: Operador
Error
Token: Tipo: ErrorLexico
Error
Token: Tipo: ErrorLexico
Token: ) Tipo: Operador
Token: ; Tipo: Operador
Error
Error
Token: scanf Tipo: Identificador
Token: ( Tipo: Operador
Error
```

## 1.2. Errores léxicos identificados



Error	Linea
♀	10
ó	10
#	0
f	2

## 2.1 Evidencia del programa usado caso de prueba #2

```
/* Programa: Hola mundo */

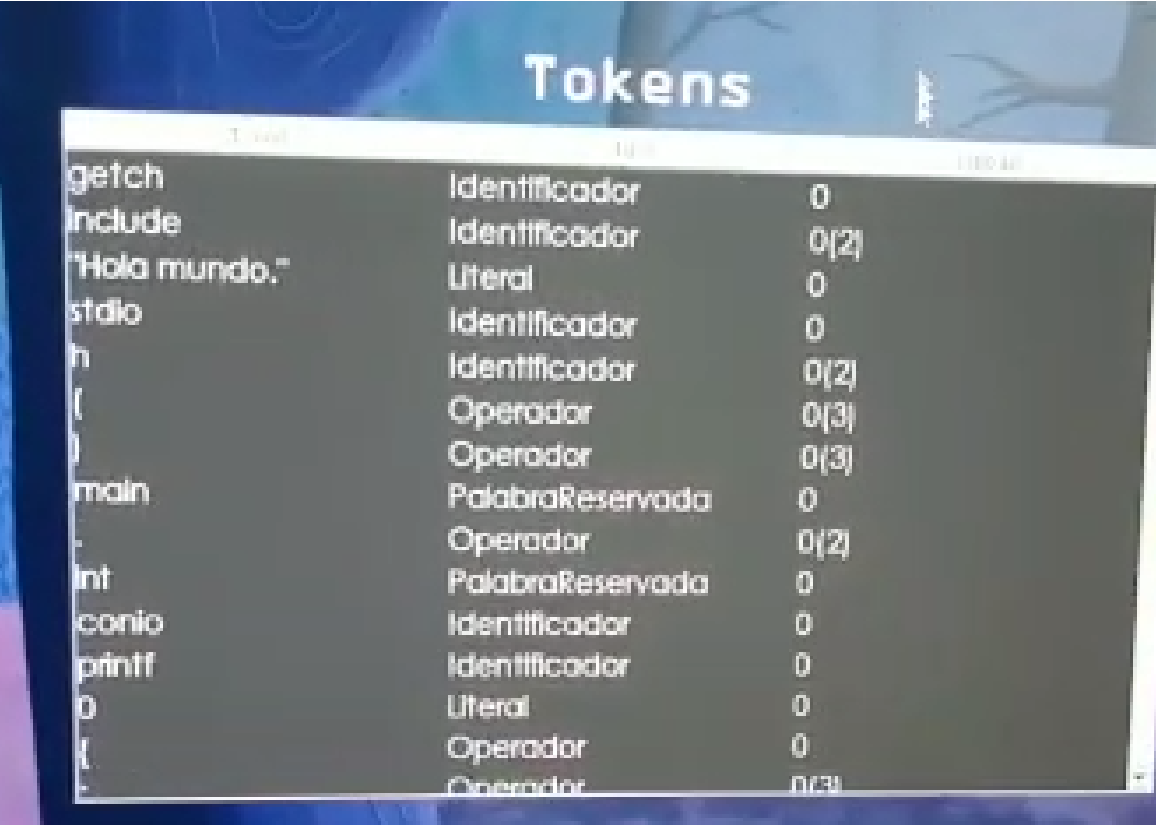
#include <conio.h>
#include <stdio.h>

int main()
{
    printf( "Hola mundo." );

    getch(); /* Pausa */

    return 0;
}
```

## 2.2 Evidencia del resultado obtenido caso de prueba #2



Token	Categoría	Cantidad
getch	Identificador	0
include	Identificador	0(2)
"Hola mundo."	Literal	0
stdio	Identificador	0
h	Identificador	0(2)
(	Operador	0(3)
)	Operador	0(3)
main	PalabraReservada	0
-	Operador	0(2)
int	PalabraReservada	0
conio	Identificador	0
printf	Identificador	0
0	Literal	0
{	Operador	0
}	Operador	0(3)