



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

**IC-5701 Compiladores e Intérpretes
Grupo 3**

**I Proyecto
Scanner**

**Profesora:
Ing. Erika Marín Schumann**

**Estudiantes:
Gilbert Rodriguez Mejías 2020061179
Diego Arturo Álvarez Esquivel 202032119
Kendall**

II Semestre del 2022

Índice

Índice	2
Introducción	3
Estrategia de solución	4
Desarrollo de las expresiones regulares	4
Análisis de resultados	6
Lecciones aprendidas	7
Casos de pruebas	8
Prueba todo correcto:	8
Código de prueba:	8
Explicación de la prueba:	9
Lista total de resultados:	9
Prueba de identificadores incorrectos	11
Código prueba	11
Explicación de la prueba	12
Lista de tokens	13
Lista de errores	15
Prueba de caracteres no admitidos	15
Código prueba	15
Explicación de la prueba:	16
Lista de tokens	17
Lista de errores	19
Manual de usuario	20
Bitácora de trabajo	22
Referencias bibliográficas	26

Introducción

Los compiladores son una de las piedras angulares cuando de desarrollo de software hablamos, pues permiten que la mayoría de los programas puedan ser implementados de una manera óptima, ya que sin ellos muchos de los lenguajes de programación que existen hoy en día no podrían funcionar. A lo largo del curso se ha planteado la importancia de los compiladores y se han investigado cada una de las partes que lo componen.

Muchas de las aplicaciones que son desarrolladas hoy en día requieren la validación de los datos de entrada, ya sea de una interfaz gráfica o un archivo. Para este fin, son necesarios los mismos conocimientos básicos que son utilizados en la construcción de la etapa de análisis léxico de un compilador, de ahí radica que sea de gran importancia tener experiencia al respecto.

La finalidad de este proyecto es poner en práctica los conocimientos y herramientas adquiridas para la etapa del análisis léxico en el curso de compiladores e intérpretes mediante la elaboración de un scanner, para el lenguaje de programación C.

El desarrollo de dicho analizador léxico se llevará a cabo mediante el lenguaje de programación Java con la ayuda de la herramienta JFlex, donde además se pondrá en práctica los conocimientos adquiridos en el curso respecto a temas referentes al análisis léxico como los autómatas finitos determinísticos y principalmente las expresiones regulares.

El scanner debe recibir un código fuente escrito en C y realizar el análisis léxico respectivo. Al finalizar el escaneo, el programa deberá desplegar al usuario el resultado del análisis Léxico efectuado. Dicho resultado comprende dos aspectos importantes, mostrar al usuario un listado de los errores léxicos y un listado con los tokens encontrados.

Estrategia de solución

Como aspecto primordial para lograr la solución del proyecto, se requirió realizar un proceso de investigación referente al uso de la herramienta JFlex, esto mediante la visualización y ejecución de ciertos tutoriales que nos permitieran conocer el adecuado funcionamiento de la herramienta así como sus características primordiales. De igual manera, realizar un estudio de la documentación oficial de JFlex nos permitió descubrir nuevas funcionalidades útiles para el desarrollo del proyecto. Con la teoría y práctica llevada a cabo en clase con respecto al tema de expresiones regulares, así como el proceso de estudio de JFlex, se procedió a la construcción de las respectivas expresiones regulares que permitieran definir el léxico del lenguaje.

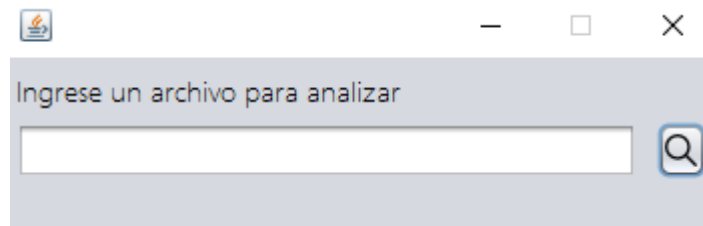
Desarrollo de las expresiones regulares

Como parte fundamental de la solución está el desarrollo de las expresiones regulares, son vitales pues permiten definir el léxico de un lenguaje, de modo que fue importante dedicar un esfuerzo adicional y poner en práctica de manera completa el conocimiento adquirido en el curso para la construcción de las mismas.

El proceso de construcción de las expresiones regulares para el proceso de permitir los números fue retador, pues fue necesario realizar un proceso de investigación de los distintos formatos aceptados por el lenguaje C para los números enteros y flotantes, llegando a la conclusión de que acepta los formatos octal, hexadecimal y flotante con exponente, que además, deben cumplir ciertas reglas para ser catalogados como tal, por ejemplo los números octales deben tener como prefijo el número 0 y además están compuestos por números en un rango del 0 al 7, y los números hexadecimales tienen el prefijo 0X o 0x y están compuestos por números en un rango del 0 al 9 y por letras de la A a la F que en hexadecimal representan los números del 10 al 15 respectivamente, de modo que es importante pensar en todas estas posibilidades en el proceso de elaboración de las expresiones que los reconozcan. Estas expresiones pueden ser encontradas en el archivo denominado Lexer.flex.

Una vez finalizado el archivo `Lexer.flex` con todas las especificaciones del lenguaje, se realiza la compilación del mismo con el objetivo de generar el archivo `Lexer.java` que cuenta con todos los métodos necesarios para revisar cada expresión regular definida en una hilera dada.

El análisis es llevado a cabo en el archivo `LexProcessor.java` que recibe la ruta de un archivo ya sea en con extensión `.txt`, `.c` o `.cpp`, este archivo es especificado por el usuario mediante la interfaz de usuario mostrada a continuación:



Desde esta interfaz se llama al método llamado `simpleAnalisis` de la clase `LexProcessor` el cual utilizará los métodos de la clase `Lexer` para extraer cada uno de los tokens contenidos en el archivo, mediante una instrucción **switch** se clasifican y separan los tokens válidos de los errores y son guardados en dos arreglos diferentes como objetos de la clase **Word**, en esta clase se guarda el token como tal, el tipo de token, la cantidad de apariciones y las líneas en las cuales aparece. Si un token que ya había sido registrado vuelve a aparecer, se actualizan las apariciones y las líneas de aparición ya existentes.

Una vez analizado todo el código, se muestra un panel con los resultados obtenidos del proceso de análisis léxico, y en caso de que haya errores, se muestra otro panel de los errores obtenidos en el proceso.

Análisis de resultados

1. Listado de errores léxicos encontrados. 100%
2. Listado de los tokens encontrados. 100%
3. Ignorar comentarios de bloque y de línea. 100%
4. Reconocer identificadores. 100%
5. Reconocer todas las palabras reservadas de C. 100%
6. Reconocer los números enteros y flotantes en los respectivos formatos aceptados por el lenguaje de programación C (Octal, Hexadecimal, Flotante con exponente). 100%
7. Reconocer los caracteres y string. 100%
8. Reconocer los operadores aceptados por C. 100%

Lecciones aprendidas

- Realizar una revisión de la documentación oficial de la herramienta JFlex nos permitió encontrar distintas funcionalidades que requerimos para el desarrollo del proyecto.
- Investigar sobre cómo el lenguaje de programación C define y acepta los distintos literales facilitó el planteamiento y desarrollo de las expresiones regulares.
- Si se presenta alguna duda, contactar a la profesora para aclararla es importante pues de este modo nos cercioramos de que el trabajo esté siendo desarrollado adecuadamente según lo solicitado y evita tener retrasos en el desarrollo por no saber si lo que se está desarrollando es correcto o no.
- Realizar una buena división de trabajo permitió desarrollar de una manera sencilla y rápida el proyecto.
- Mantener una comunicación constante para saber si alguno de los integrantes tiene problemas para terminar o realizar la parte de trabajo que le corresponde permitió el desarrollo rápido y correcto de trabajo.

Casos de pruebas

Prueba todo correcto:

Código de prueba:

```
#include <stdio.h>
#include <stdlib.h>
/*
Esto es comentario
de bloque
*/
int main(void)
{
    int x,y,z,cont;

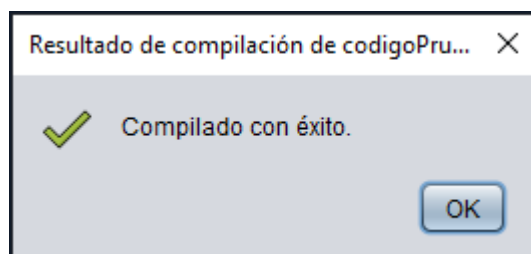
    x=0;
    y=1;

    printf("0\n1\n",z);
    // Comentarios en mitad
    for (cont=1;cont<=20;cont=cont+1)
    {
        z=x+y;
        printf("%d\n",z);
        x=y;
        y=z;
    }

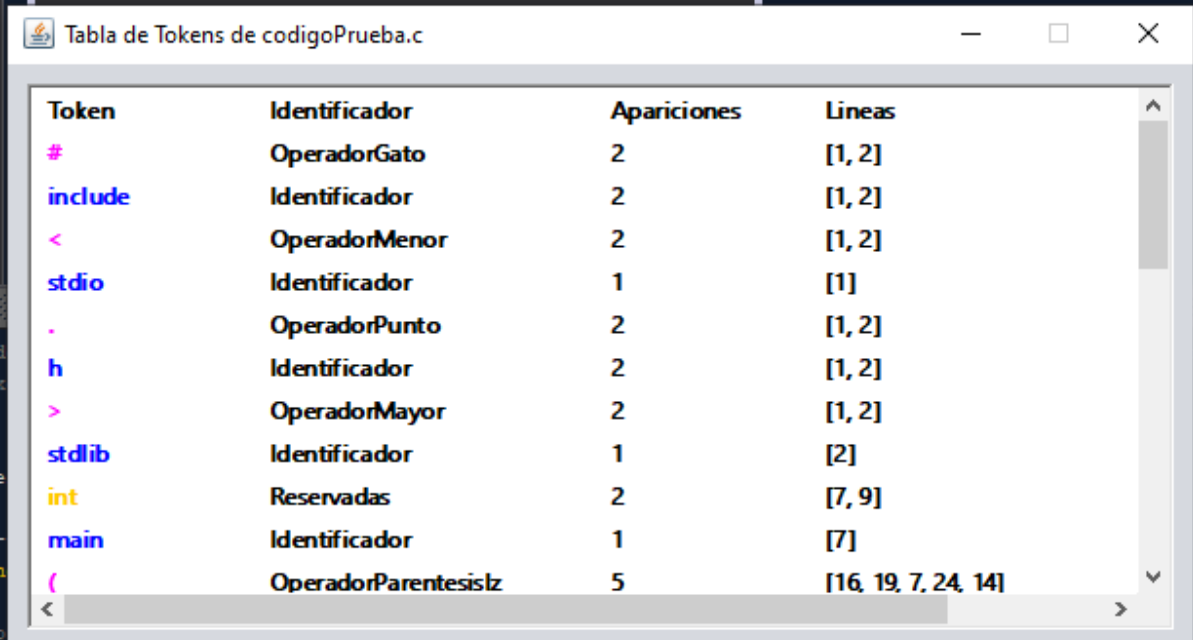
    system("PAUSE");
    return 0;
    //Esto es un comentario el final
}
```

Explicación de la prueba:

Después de indicarle al analizador cuál archivo va a cargar, para efectos de esta prueba, se cargó el archivo que contiene el código anterior, como resultado, indicó con la siguiente ventana que todo el código estaba bien y no se encontraron errores.



Después aparece una ventana emergente, la cual se verá a continuación, en donde se muestran los tokens en el orden en el que aparecen con su respectivo identificador (Nombre para representar el token), la cantidad de apariciones que tienen en el código y las líneas en que aparecen (En esta parte las líneas aparecerán entre [] y separadas por coma pero si en una misma línea aparece más de una vez el mismo token, se pondrá el número de la línea y entre paréntesis el número de veces que se repite)



Token	Identificador	Apariciones	Lineas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	2	[1, 2]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[7, 9]
main	Identificador	1	[7]
(OperadorParentesisIz	5	[16, 19, 7, 24, 14]

Lista total de resultados:

Token	Identificador	Apariciones	Líneas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	2	[1, 2]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[7, 9]
main	Identificador	1	[7]
(OperadorParentesisIz	5	[16, 19, 7, 24, 14]
void	Reservadas	1	[7]
)	OperadorParentesisDer	5	[16, 19, 7, 24, 14]
{	OperadorCorcheteIz	2	[17, 8]
x	Identificador	4	[18, 19, 9(2)]
,	OperadorComa	5	[19, 9(3), 14]
y	Identificador	5	[18, 20(2), 9, 11]
z	Identificador	5	[17, 19, 21, 9, 14]
cont	Identificador	5	[16(4), 9]

;	OperadorPuntoComa	12	[16(2), 18, 19, 20, 21, 24, 9, 25, 11, 12, 14]
=	OperadorIgual	7	[16(2), 18, 20, 21, 11, 12]
0	Entero	2	[25, 11]
1	Entero	3	[16(2), 12]
printf	Identificador	2	[18, 12]
"0\n1\n"	Hilera	1	[14]
for	Reservadas	1	[16]
<=	OperadorMenorIgual	1	[16]
20	Entero	1	[16]
+	OperadorSuma	2	[16, 18]
"%d\n"	Hilera	1	[19]
}	OperadorCorcheteDer	2	[22, 27]
system	Identificador	1	[22]
"PAUSE"	Hilera	1	[24]
return	Reservadas	1	[25]

Prueba de identificadores incorrectos

Código prueba

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int num1, num2, num3, num4, num5, num6, num7, num8, num9, num10, 11num;
    char char1, char2, char3, 4tochar;

    num1 = 29;
    num2 = 050;
    num3 = 0x1A2;

    num4 = -12.2303;
    num5 = 012.56;
    num6 = 0X4F.2p0;
    num7 = 0.96;
    num8 = 2.15E2;
    num9 = 5e-10;
    num10 = .0007E4;
    11num = -5.4e-12;

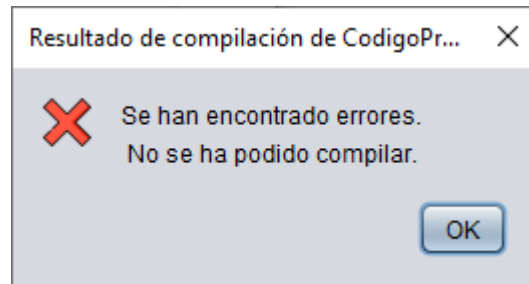
    char1 = 'k';
    char2 = '4';
    char3 = '\n';
    4tochar = 'c';

    char hilera1[] = "Esto es una hilera";
    char hilera2[] = "Hola \"Mundo\"";
    char hilera3[] = "k";
    char 4tahilera[] = "Hola niño \n Cómo está?";
    char hilera5[] = "Welcome\nTo\nGeeks\tFor\tGeeks";
    char hilera6[] = "Diego\n\tArturo\t Álvarez";

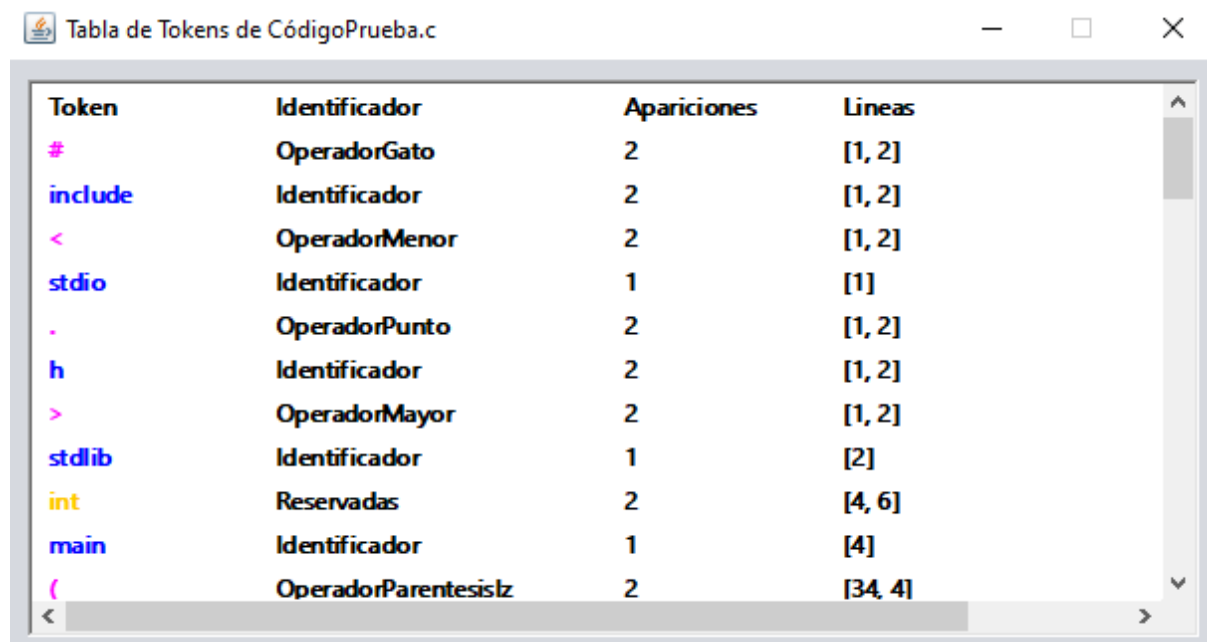
    system("PAUSE");
    return 0;
}
```

Explicación de la prueba

Después de indicarle al analizador cuál archivo va a cargar, para efectos de esta prueba, se cargó el archivo que contiene el código anterior, como resultado, indicó con la siguiente ventana que el código presentaba errores y no fue compilado.



Después aparece una ventana emergente, la cual se verá a continuación, en donde se muestran los tokens en el orden en el que aparecen con su respectivo identificador (Nombre para representar el token), la cantidad de apariciones que tienen en el código y las líneas en que aparecen (En esta parte las líneas aparecerán entre [] y separadas por coma, pero si en una misma línea aparece más de una vez el mismo token, se pondrá el número de la línea y entre paréntesis el número de veces que se repite).



Token	Identificador	Apariciones	Lineas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	2	[1, 2]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[4, 6]
main	Identificador	1	[4]
(OperadorParentesisIz	2	[3, 4]

Finalmente se muestra una ventana con el mismo formato de la anterior que muestra cada uno de los errores encontrados en el código, los errores que en esta prueba buscamos atacar, fueron los de identificadores que no fuesen válidos en el lenguaje C.

Token	Identificador	Apariciones	Lineas
11num	ERROR	2	[19, 6]
4tochar	ERROR	2	[7, 24]
4tahilera	ERROR	1	[30]

Lista de tokens

Token	Identificador	Apariciones	Líneas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	2	[1, 2]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[4, 6]
main	Identificador	1	[4]
(OperadorParentesisIz	2	[34, 4]
void	Reservadas	1	[4]
)	OperadorParentesisDer	2	[34, 4]
{	OperadorCorcheteIz	1	[5]
num1	Identificador	2	[6, 7]
,	OperadorComa	13	[6(10), 7(3)]
num2	Identificador	2	[6, 9]
num3	Identificador	2	[6, 10]
num4	Identificador	2	[6, 11]
num5	Identificador	2	[6, 13]
num6	Identificador	2	[6, 14]
num7	Identificador	2	[6, 15]
num8	Identificador	2	[16, 6]
num9	Identificador	2	[17, 6]

num10	Identificador	2	[18, 6]
;	OperadorPuntoComa	25	[6, 7, 9, 10, 11, 13, 14,
			15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 34, 35]
char	Reservadas	7	[32, 7, 27, 28, 29, 30, 31]
char1	Identificador	2	[20, 7]
char2	Identificador	2	[22, 7]
char3	Identificador	2	[7, 23]
=	OperadorIgual	21	[32, 9, 10, 11, 13, 14, 15,
			16, 17, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 30, 31]
29	Entero	1	[9]
050	Octal	1	[10]
0x1A2	Hexadecimal	1	[11]
-12.2303	Flotante	1	[13]
012.56	OctalFlotante	1	[14]
0X4F.2p0	HexadecimalFlotante	1	[15]
0.96	Flotante	1	[16]
2.15E2	FlotanteExponente	1	[17]
5e-10	FlotanteExponente	1	[18]
.0007E4	FlotanteExponente	1	[19]
-5.4e-12	FlotanteExponente	1	[20]
'k'	Caracter	1	[22]
'4'	Caracter	1	[23]
'\n'	Caracter	1	[24]
'c'	Caracter	1	[25]
hilera1	Identificador	1	[27]
[OperadorParCuadradoIz	6	[32, 27, 28, 29, 30, 31]
]	OperadorParCuadradoDer	6	[32, 27, 28, 29, 30, 31]
"Esto es una hilera"	Hilera	1	[27]
hilera2	Identificador	1	[28]
"Hola \"Mundo\""	Hilera	1	[28]
hilera3	Identificador	1	[29]
"k"	Hilera	1	[29]
"Hola niño \n Cómo está?"	Hilera	1	[30]
hilera5	Identificador	1	[31]
"Welcome\nTo\nGeeks\tFor\tGeeks"	Hilera	1	[31]
hilera6	Identificador	1	[32]
"Diego\n\tArturo\t Álvarez"	Hilera	1	[32]
system	Identificador	1	[32]
"PAUSE"	Hilera	1	[34]
return	Reservadas	1	[35]
0	Entero	1	[35]
}	OperadorCorcheteDer	1	[36]

Lista de errores

Token	Identificado	Apariciones	Líneas
11num	ERROR	2	[19, 6]
4tochar	ERROR	2	[7, 24]
4tahilera	ERROR	1	[30]

Prueba de caracteres no admitidos

Código prueba

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int ñum1, num2, num3, num4, num5, num6, num7, num8, num9, num10, 11num;
    char char1, char2, char3, char4;

    num1 = 29;
    num2 = 050;
    num3 = 0x1A2;

    num4 = -12.2303;
    num5 = 012.56;
    num6 = 0X4F.2p0;
    num7 = 0.96;
    num8 = 2.15E2;
    num9 = 5e-10;
    num10 = .0007E4;
    11num = -5.4e-12;

    char1 = 'k';
    char2 = '4';
    char3 = '\n';
    tochar = 'c';

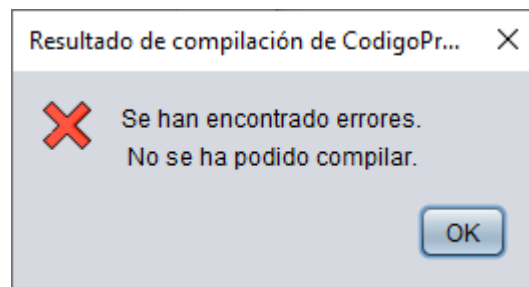
    char hilera1[] = "Esto es una hilera";
    char hilera2[] = "Hola \"Mundo\"";
    char hilera3[] = "k";
    char hilera[] = "Hola niño \n Cómo está?";
    char hilera5[] = "Welcome\nTo\nGeeks\tFor\tGeeks";
    char hülera6[] = "Diego\n\tArturo\t Álvarez";

    system("PAUSE");
    return 0;
```

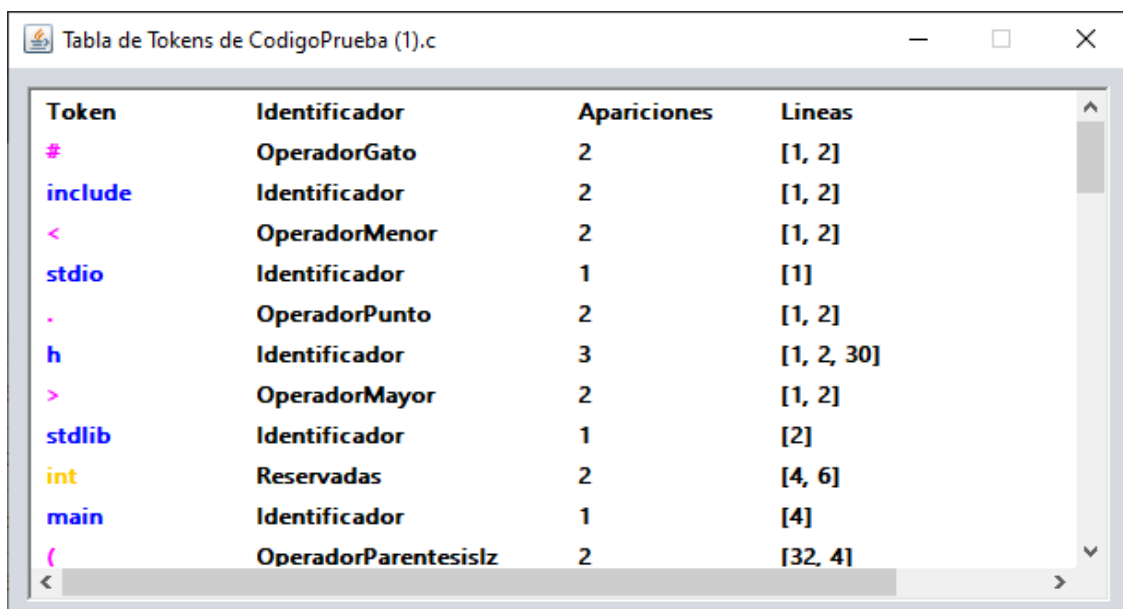
```
}
```

Explicación de la prueba:

Después de indicarle al analizador cuál archivo va a cargar, para efectos de esta prueba, se cargó el archivo que contiene el código anterior, como resultado, indicó con la siguiente ventana que el código presentaba errores y no fue compilado.



Después aparece una ventana emergente, la cual se verá a continuación, en donde se muestran los tokens en el orden en el que aparecen con su respectivo identificador (Nombre para representar el token), la cantidad de apariciones que tienen en el código y las líneas en que aparecen (En esta parte las líneas aparecerán entre [] y separadas por coma, pero si en una misma línea aparece más de una vez el mismo token, se pondrá el número de la línea y entre paréntesis el número de veces que se repite).

Una ventana emergente con el título "Tabla de Tokens de CodigoPrueba (1).c". Contiene una tabla con cuatro columnas: Token, Identificador, Apariciones y Lineas. La tabla muestra los tokens encontrados en el código, como #, include, <, stdio, ., h, >, stdlib, int, main, (, y sus respectivos identificadores, apariciones y líneas.

Finalmente se muestra una ventana con el mismo formato de la anterior que muestra cada uno de los errores encontrados en el código, los errores que en esta prueba buscamos atacar, fueron los de caracteres inválidos en el lenguaje C.

Tabla de Errores de CodigoPrueba (1).c			
Token	Identificador	Apariciones	Lineas
ñ	ERROR	1	[6]
é	ERROR	1	[25]
ü	ERROR	1	[30]

Lista de tokens

Token	Identificador	Apariciones	Líneas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	3	[1, 2, 30]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[4, 6]
main	Identificador	1	[4]
(OperadorParentesisIz	2	[32, 4]
void	Reservadas	1	[4]
)	OperadorParentesisDer	2	[32, 4]
{	OperadorCorcheteIz	1	[5]
um1	Identificador	1	[6]
,	OperadorComa	13	[6(10), 7(3)]
num2	Identificador	2	[6, 9]
num3	Identificador	2	[6, 10]
num4	Identificador	2	[6, 11]
num5	Identificador	2	[6, 13]
num6	Identificador	2	[6, 14]
num7	Identificador	2	[6, 15]
num8	Identificador	2	[16, 6]
num9	Identificador	2	[17, 6]
num10	Identificador	2	[18, 6]
num11	Identificador	1	[6]

;	OperadorPuntoComa	23	[32, 33, 6, 7, 9, 10, 11, 13, 14,
15, 16, 17, 18, 19, 21, 22, 23, 25, 26, 27, 28, 29, 30]			
char	Reservadas	7	[7, 25, 26, 27, 28, 29, 30]
char1	Identificador	2	[19, 7]
char2	Identificador	2	[21, 7]
char3	Identificador	2	[22, 7]
char4	Identificador	1	[7]
num1	Identificador	1	[7]
=	OperadorIgual	19	[9, 10, 11, 13, 14, 15, 16, 17, 18,
19, 21, 22, 23, 25, 26, 27, 28, 29, 30]			
29	Entero	1	[9]
050	Octal	1	[10]
0x1A2	Hexadecimal	1	[11]
-12.2303	Flotante	1	[13]
012.56	OctalFlotante	1	[14]
0X4F.2p0	HexadecimalFlotante	1	[15]
0.96	Flotante	1	[16]
2.15E2	FlotanteExponente	1	[17]
5e-10	FlotanteExponente	1	[18]
.0007E4	FlotanteExponente	1	[19]
'k'	Caracter	1	[21]
'4'	Caracter	1	[22]
'\n'	Caracter	1	[23]
hil	Identificador	1	[25]
ra1	Identificador	1	[25]
[OperadorParCuadradoIz	6	[25, 26, 27, 28, 29, 30]
]	OperadorParCuadradoDer	6	[25, 26, 27, 28, 29, 30]
"Esto es una hilera"	Hilera	1	[25]
hilera2	Identificador	1	[26]
"Hola \"Mundo\""	Hilera	1	[26]
hilera3	Identificador	1	[27]
"k"	Hilera	1	[27]
hilera	Identificador	1	[28]
"Hola niño \n Cómo está?"	Hilera	1	[28]
hilera5	Identificador	1	[29]
"Welcome\nTo\nGeeks\tFor\tGeeks"	Hilera	1	[29]
lera6	Identificador	1	[30]
"Diego\n\tArturo\t Álvarez"	Hilera	1	[30]
system	Identificador	1	[30]
"PAUSE"	Hilera	1	[32]
return	Reservadas	1	[33]
0	Entero	1	[33]
}	OperadorCorcheteDer	1	[34]

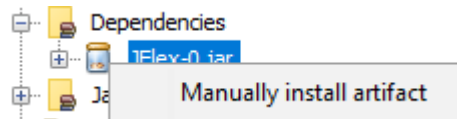
Lista de errores

Token	Identificador	Apariciones	Líneas
ñ	ERROR	1	[6]
é	ERROR	1	[25]
ü	ERROR	1	[30]

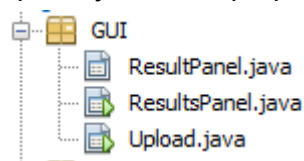
Manual de usuario


Para utilizar el programa se debe contar con Netbeans, Java 8 o superior y el archivo [Jflex.jar](#) adjunto.

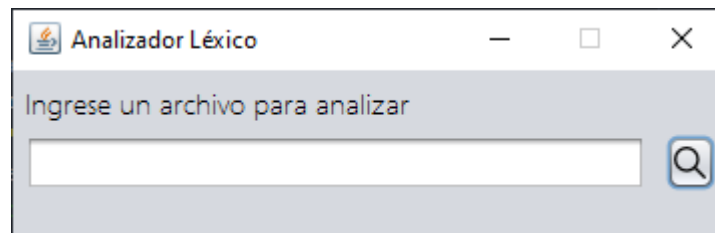
El archivo Jflex.jar se debe instalar en las dependencias del proyecto para poderse utilizar



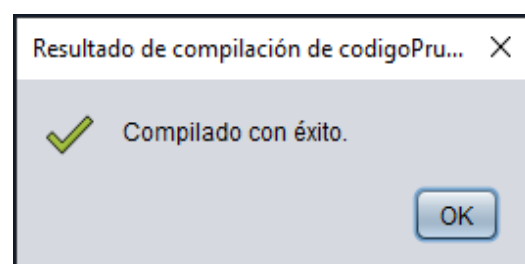
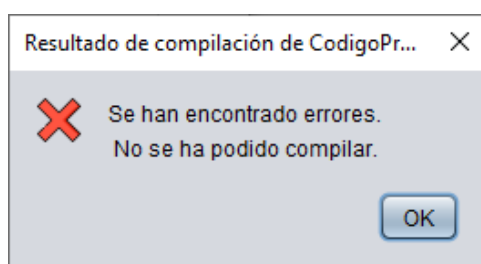
Una vez instalado el archivo Jflex.jar se puede correr el proyecto ejecutando el archivo Upload.java en el paquete GUI



Se mostrará una ventana donde a través del botón marcado con una lupa  se abrirá un navegador de archivos. Aquí podrás seleccionar un archivo con la extensión .c, .cpp o .txt.

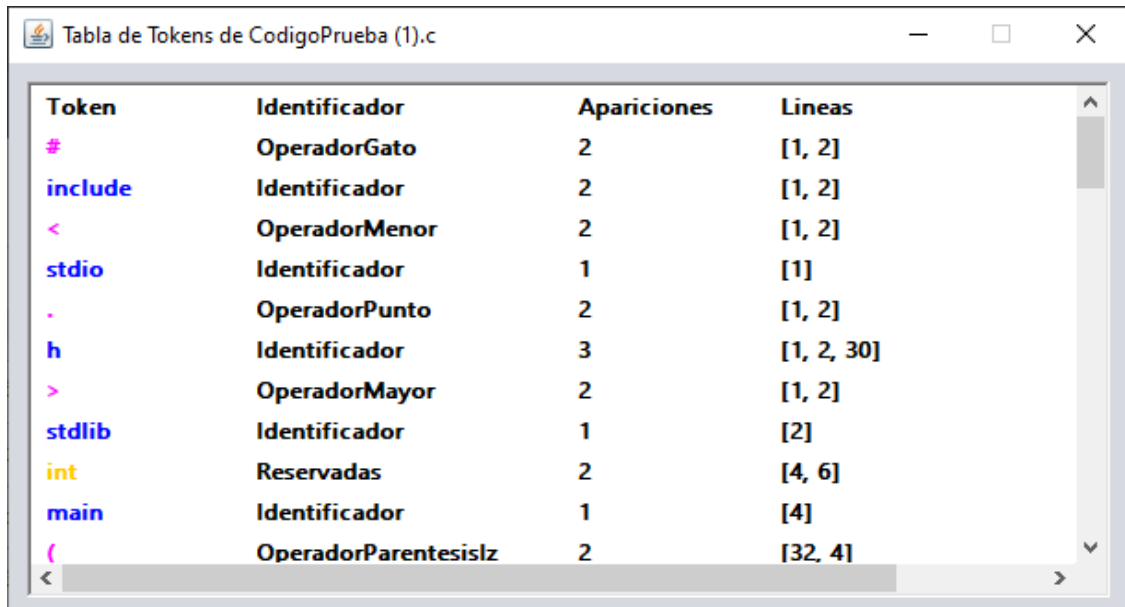


Una vez el archivo sea seleccionado el scanner funcionará y se mostrará una ventana con alguno de los dos mensajes mostrados a continuación:



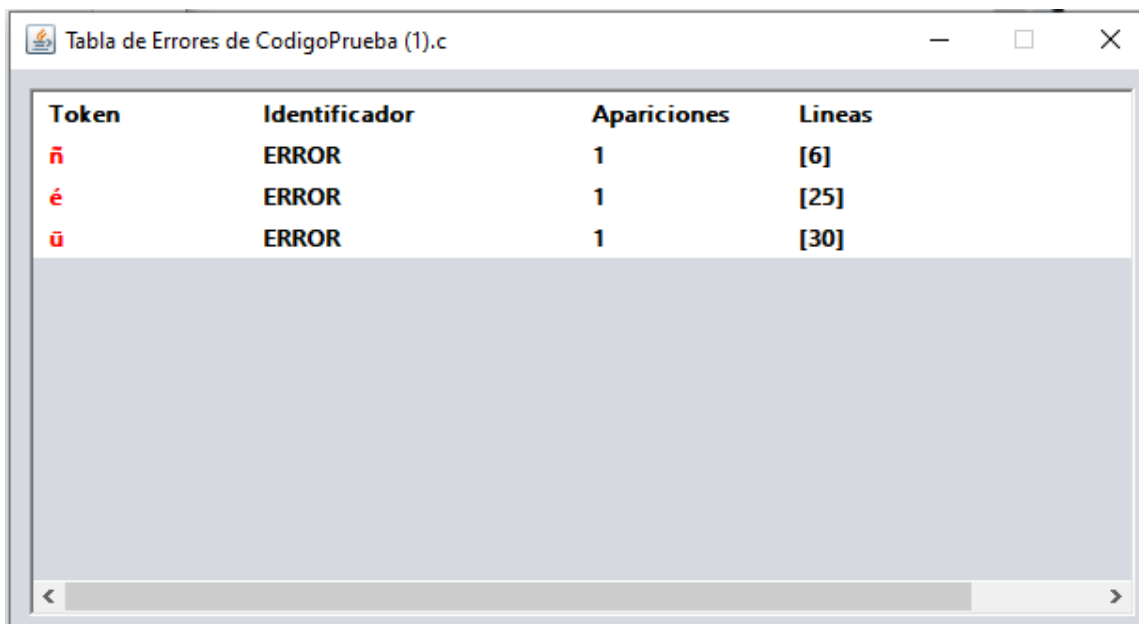
Si el programa presenta algún error se mostrará la primera imagen, de lo contrario, se mostrará la segunda.

Como resultado, se mostrará una ventana emergente, la cual se verá a continuación, en donde se muestran los tokens en el orden en el que aparecen con su respectivo identificador (Nombre para representar el token), la cantidad de apariciones que tienen en el código y las líneas en que aparecen (En esta parte las líneas aparecerán entre [] y separadas por coma, pero si en una misma línea aparece más de una vez el mismo token, se pondrá el número de la línea y entre paréntesis el número de veces que se repite).



Token	Identificador	Apariciones	Lineas
#	OperadorGato	2	[1, 2]
include	Identificador	2	[1, 2]
<	OperadorMenor	2	[1, 2]
stdio	Identificador	1	[1]
.	OperadorPunto	2	[1, 2]
h	Identificador	3	[1, 2, 30]
>	OperadorMayor	2	[1, 2]
stdlib	Identificador	1	[2]
int	Reservadas	2	[4, 6]
main	Identificador	1	[4]
(OperadorParentesisIz	2	[32, 4]

Finalmente en caso de que se hayan encontrado errores, se muestra una ventana adicional con el mismo formato de la anterior que muestra cada uno de los errores encontrados en el código.



Token	Identificador	Apariciones	Lineas
ñ	ERROR	1	[6]
é	ERROR	1	[25]
ü	ERROR	1	[30]

El enlace del repositorio del proyecto se adjunta [aquí](#).

Bitácora de trabajo

Minuta de Reunión #1			
Fecha	02/09/2022.	Hora de inicio	8:00 P.M.
Lugar	Discord.	Hora de fin	9:00 P.M.
Objetivos			
Crear y compartir el proyecto mediante Apache Netbeans.			
Asistencia			
Nombre	Puesto	Asistencia	Firma o Motivo
Diego Arturo Álvarez Esquivel.		Sí	
Kendall Cascante Mesén		Sí	Kendall C
Gilbert Rodríguez Mejías.		Sí	Diego Álvarez E.

Asuntos tratados:

- **Asuntos con prioridad:**
 - Creación del proyecto y agregar lo necesario para el funcionamiento de la herramienta JFlex.
- **Asuntos secundarios:**
 - División del trabajo por realizar.

Compromisos Asumidos			
No	Tarea	Responsable	Fecha de entrega
1	Probar la herramienta JFlex y revisar la documentación respectiva.	Todos.	Jueves 6 de septiembre del 2022.

Minuta de Reunión #2			
Fecha	06/09/2022.	Hora de inicio	3:00 P.M.
Lugar	Discord.	Hora de fin	4:00 P.M.
Objetivos			
Dividir el trabajo por realizar.			
Asistencia			
Nombre	Puesto	Asistencia	Firma o Motivo
Diego Arturo Álvarez Esquivel.		Sí	
Kendall Cascante Mesén		Sí	Kendall C
Gilbert Rodríguez Mejías.		Sí	Diego Álvarez E.

Asuntos tratados:

- **Asuntos con prioridad:**
 - División del trabajo por realizar.
- **Asuntos secundarios:**
 - No hay.

Compromisos Asumidos			
No	Tarea	Responsable	Fecha de entrega
1	Diego: GUI, mitad números. Kendall: expresiones regulares carácter, string, mitad números. Gil: operadores, investigar tablas y comentarios	Todos.	Lunes 12 de septiembre del 2022.

Minuta de Reunión #3			
Fecha	12/09/2022.	Hora de inicio	7:00 P.M.
Lugar	Discord.	Hora de fin	7:30 P.M.
Objetivos			
Revisar los avances realizados e iniciar la documentación del trabajo.			
Asistencia			
Nombre	Puesto	Asistencia	Firma o Motivo
Diego Arturo Álvarez Esquivel.		Sí	
Kendall Cascante Mesén		Sí	Kendall C
Diego Arturo Álvarez Esquivel.		Sí	Diego Álvarez E.

Asuntos tratados:

- **Asuntos con prioridad:**
 - Empezar la documentación y verificar lo avanzado.
- **Asuntos secundarios:**
 - No hay

Compromisos Asumidos			
No	Tarea	Responsable	Fecha de entrega
1	Probar y mejorar la interfaz gráfica.	Todos.	Miércoles 14 de septiembre del 2022.

Minuta de Reunión #4			
Fecha	14/09/2022.	Hora de inicio	8:30 P.M.
Lugar	Discord.	Hora de fin	9:00 P.M.
Objetivos			
Dividir la documentación y revisar el código fuente.			
Asistencia			
Nombre	Puesto	Asistencia	Firma o Motivo
Diego Arturo Álvarez Esquivel.		Sí	
Kendall Cascante Mesén		Sí	Kendall
Diego Arturo Álvarez Esquivel.		Sí	Diego Álvarez E.

Asuntos tratados:

- **Asuntos con prioridad:**
 - División de la documentación y revisión del código.
- **Asuntos secundarios:**
 - Preguntas para la profesora.

Compromisos Asumidos			
No	Tarea	Responsable	Fecha de entrega
1	Gilberth: análisis de resultados y lecciones aprendidas. Diego: introducción y estrategia de la solución. Kendall: revisión de problemas con los errores, detalle de construcción de ER de números y literales en la estrategia de solución.	Todos.	Martes 20 de setiembre del 2022.

Referencias bibliográficas

- [1] <https://jflex.de/manual.html#Example>
- [2] <http://ejercicioscpp.blogspot.com/2012/09/literal-en-c.html#:~:text=Un%20literal%20es%20cualquier%20valor,car%C3%A1cter%20y%20cadena%20de%20caracteres>
- [3] <https://baulderasec.wordpress.com/programacion/cc/elementos-del-lenguaje-c/literales/literales-de-un-solo-caracter/>
- [4] <https://www.cs.auckland.ac.nz/courses/compsci330s1c/lectures/330ChaptersPDF/Chapt1.pdf>
- [5] http://westes.github.io/flex/manual/How-can-I-match-C_002dstyle-comments_003f.html
- [6] https://www.it.uc3m.es/pbasanta/asng/course_notes/data_types_es.html#data_types_characters_and_strings
- [7] <https://www.includehelp.com/c/working-with-octal-values-in-c-programming-language.aspx?ref=rp>
- [8] <https://www.includehelp.com/c/working-with-hexadecimal-values-in-c-programming-language.aspx>
- [9] <https://www.ibm.com/docs/es/developer-for-zos/14.2.0?topic=programs-c-reserved-keywords>
- [10] <https://www.ibm.com/docs/es/developer-for-zos/14.2.0?topic=programs-c-operators-operands>
- [11] <http://books.gigatux.nl/mirror/cinanutshell/0596006977/cinanut-CHP-3-SECT-2.html>
- [12] https://en.cppreference.com/w/c/language/floating_constant
- [13] <https://www.exploringbinary.com/hexadecimal-floating-point-constants/>
- [14] https://www.ibm.com/docs/en/zos/2.2.0?topic=SSLTBW_2.2.0/com.ibm.zos.v2r2.asma400/asmr1021142.htm
- [15] <https://www.ibm.com/docs/en/rdfi/9.6.0?topic=tokens-literals>
- [16] <https://learn.microsoft.com/es-es/cpp/c-language/c-string-literals?view=msvc-170>