

**UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA**

**Facultad:** Ingeniería en Sistemas

**Nombre:** Jairo Isaí Flores Campos

**Carnet:** 2890-19-25336

**Nombre:** José Emilio Com Alfaro

**Carnet:** 2890-19-27573

**Nombre:** Abner Evelio Vicente Perico

**Carnet:** 2890-19-13417

**Nombre:** Esdras Estuardo Hernández y Hernández

**Carnet:** 2890-19-1095

**Curso:** Compiladores

**Docente:** Ing. Karen Patricia Hernández Rivera

**Tarea:** proyecto final.

# Introducción

Las herramientas de flex y bison son de gran ayuda al momento de ir haciendo la estructura y el orden de lectura de los analizadore léxicos. Por ejemplo, la estructura de un int variable = 15; se pude decir que tiene palabra reservada = (int), identificador (Letras), signoigual (=), y fin de línea (;). Mientras que en bison definimos todos los tokens de entrada, si por ejemplo tenemos un token de TOK\_NUMBER, al momento de que en flex detectemos que es de este tipo pues podemos hacer llamada a este token, y así sucesivamente ir acompañando tanto flex como bison en el analizador léxico.

# Índice

Contenido

[Introducción 2](#_Toc105010251)

[Índice 3](#_Toc105010252)

[Herramientas Utilizadas 4](#_Toc105010253)

[Introducción a Flex 4](#_Toc105010254)

[Interacción con Bison 6](#_Toc105010255)

[Pruebas de software 10](#_Toc105010256)

[Código Fuente 12](#_Toc105010257)

[Conclusiones 12](#_Toc105010258)

[Bibliografía 12](#_Toc105010259)

# Herramientas Utilizadas

Archivo Flex

Archivo de entrada.txt

Archivo de salida.txt

Archivo ejecutable ProyectoFinal.exe

# Introducción a Flex

Flex es una una herramienta que permite generar analizadores léxicos. A partir de un

conjunto de expresiones regulares, Flex busca concordancias en un fichero de entrada y

ejecuta acciones asociadas a estas expresiones. Es compatible casi al 100% con Lex,

una herramienta clásica de Unix para la generación de analizadores léxicos, pero es un

desarrollo diferente realizado por GNU bajo licencia GPL.

Los ficheros de entrada de Flex (normalmente con la extensión .l) siguen el siguiente

esquema:

%%

patrón1 {acción1}

patrón2 {acción2}

...

donde:

patrón: expresión regular

acción: código C con las acciones a ejecutar cuando se encuentre concordancia del

patrón con el texto de entrada

Flex recorre la entrada hasta que encuentra una concordancia y ejecuta el código

asociado. El texto que no concuerda con ningún patrón lo copia tal cual a la salida. Por

ejemplo:

%%

a\*b {printf(“X”);};

re ;

El ejecutable correspondiente transforma el texto:

abre la puertaab

en

X la puertX

pues ha escrito X cada vez que ha encontrado ab o aab y nada cuando ha encontrado re.

Flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado

ningún nombre de fichero, con la descripción de un escáner a generar. La descripción se

encuentra en forma de parejas de expresiones regulares y código C, denominadas

reglas. Flex genera como salida un fichero fuente en C, ‘lex.yy.c’, que define una

función ‘yylex()’. Este fichero se compila y se enlaza con la librería de Flex para

producir un ejecutable. Cuando se arranca el fichero ejecutable, este analiza su entrada

en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el

código C correspondiente.

El fichero de entrada de Flex está compuesto de tres secciones, separadas por una línea

donde aparece únicamente un ‘%%’ en esta:

definiciones

%%

reglas

%%

código de usuario

3

Introducción a Flex y Bison

La sección de definiciones contiene declaraciones de definiciones de nombres sencillas

para simplificar la especificación del escáner, y declaraciones de condiciones de

arranque, que se explicarán en una sección posterior. Las definiciones de nombre tienen

la forma:

nombre definición

El "nombre" es una palabra que comienza con una letra o un subrayado (‘\_’) seguido

por cero o más letras, dígitos, ‘\_’, o ‘-’ (guión). La definición se considera que

comienza en el primer carácter que no sea un espacio en blanco siguiendo al nombre y

continuando hasta el final de la línea. Posteriormente se puede hacer referencia a la

definición utilizando "{nombre}", que se expandirá a "(definición)". Por ejemplo,

DIGITO [0-9]

ID [a-z][a-z0-9]\*

define "DIGITO" como una expresión regular que empareja un dígito sencillo, e "ID"

como una expresión regular que empareja una letra seguida por cero o más letras o

dígitos. Una referencia posterior a

{DIGITO}+"."{DIGITO}\*

es idéntica a

([0-9])+"."([0-9])\*

y empareja uno o más dígitos seguido por un ‘.’ seguido por cero o más dígitos.

La sección de reglas en la entrada de Flex contiene una serie de reglas de la forma:

patrón acción

donde el patrón debe estar sin sangrar y la acción debe comenzar en la misma línea.

Finalmente, la sección de código de usuario simplemente se copia a ‘lex.yy.c’

literalmente. Esta sección se utiliza para rutinas de complemento que llaman al escáner

o son llamadas por este. La presencia de esta sección es opcional; Si se omite, el

segundo ‘%%’ en el fichero de entrada se podría omitir también.

Patrones

Los patrones en la entrada se escriben utilizando un conjunto extendido de expresiones

regulares y usando como alfabeto cualquier carácter ASCII. Cualquier símbolo excepto

el espacio en blanco, tabulador, cambio de línea y los caracteres especiales se escriben

tal cual en las expresiones regulares (patrones) de Flex. Los caracteres especiales son:

“ \ [ ^ - ? . \* + | ( ) $ / { } % < >

Algunos de los patrones de Flex son:

x

empareja el carácter ‘x’

.

cualquier carácter excepto una línea nueva

[xyz]

un conjunto de caracteres; en este caso, el patrón empareja una ‘x’, una ‘y’, o una ‘z’

[abj-oZ]

un conjunto de caracteres con un rango; empareja una ‘a’, una ‘b’, cualquier letra desde la

‘j’ hasta la ‘o’, o una ‘Z’

[^A-Z]

# Interacción con Bison

Uno de los usos principales de Flex es como acompañante del analizador de gramáticas

Bison (o de Yacc). Los analizadores Bison necesitan una función llamda ‘yylex()’ para

devolverles el siguiente token de la entrada. Esa función devuelve el tipo del próximo

token y además puede poner cualquier valor asociado en la variable global yylval. Para

usar Flex con Bison, normalmente se especifica la opción –d de Bison para que genera

el fichero ‘y.tab.h’ que contiene las definiciones de todos los ‘%tokens’ que aparecen el

fuente Bison. Este fichero de cabecera se incluye después en el fuente de Flex. Por

ejemplo, si uno de los tokens es “TOK\_NUMBER”, parte del fichero Flex podría ser:

%{

#include "y.tab.h"

%}

%%

[0-9]+ yylval = atoi( yytext ); return TOK\_NUMBER;

Compilación y ejecución de un programa Flex

Al ejecutar el comando flex nombre\_fichero\_fuente se creará un fichero en C llamado

“lex.yy.c”. Si se compila este fichero con la instrucción “gcc lex.yy.c –lfl –o

nombre\_ejecutable” (-lfl indica enlazar con la biblioteca de flex) se obtendrá como resultado un

fichero ejecutable llamado nombre\_ejecutable).

Una vez se ha creado el fichero ejecutable se puede ejecutar directamente. Flex esperará que se introduzca

texto por la entrada estándar (teclado) y lo analizará cada vez que se pulse el retorno de carro. Para

terminar con el análisis normalmente hay que pulsar <CTRL>-D. Para poder probarlo con entradas más

largas, lo más sencillo es crear archivos de texto y usar redirecciones para que el ejecutable lea estos

ficheros como entrada a analizar. Por ejemplo si hemos creado un analizador llamado prueba1 y un

fichero de texto con datos para su análisis, llamado entrada.txt, podemos ejecutar $prueba1 <

entrada.txt.

Si se quiere ejecutar algún código al final de un análisis de Flex (para mostrar resultados por ejemplo) hay

al menos dos opciones:

%%

reglas...

C++ es considerado un C más grande y potente. La sintaxis de C++ es una extensión de C, al que se han añadido numerosas propiedades, fundamentalmente orientadas a objetos. C ANSI 1 ya adoptó numerosas características de C++, por lo que la emigración de C a C++ no suele ser difícil. En este apéndice se muestran las reglas de sintaxis del estándar clásico de C++ recogidas en al Annotated Reference Manual (ARM), de Stroustrup & Ellis, así como las últimas propuestas incorporadas al nuevo borrador de C++ ANSI, que se incluyen en las versiones 3.0 (actual) y 4.0 (futura) de AT&T C++

Un programa en C++ es una secuencia de caracteres que se agrupan en componentes léxicos (tokens) que comprenden el vocabulario básico del lenguaje. Estos componentes de léxico son: palabras reservadas, identificadores, constantes, constantes de cadena, operadores y signos de puntuación

Los caracteres que se pueden utilizar para construir elementos del lenguaje (componentes léxicos o tokens) son:

a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 + - \* / = ( ) { } [ ] < > ´ ” ! @ =/ $ ^ & % - : . , ; ? \ | caracteres espacio (blancos y tabulaciones).



La sintaxis de JavaScript define un conjunto de reglas que deben seguirse para escribir código correctamente, y darle así instrucciones al programa en el que estemos trabajando, ejecutando la tarea especificada de dicho código.

Veámoslo de esta manera, en nuestro lenguaje el español, si expresamos la siguiente oración:

* Resfriado me habría la lluvia mojado con me si hubiera.

¿Se entiende? la verdad ¡No!, ya que las palabras se encuentran en desorden, la sintaxis correcta de esta oración sería:

* Si me hubiera mojado con la lluvia, me habría resfriado.

Como te das cuenta, se necesita un orden correcto, del sujeto y predicado, para poder expresar lo correcto. Pues nuestro lenguaje tiene una serie de reglas establecidas por la sintaxis para la elaboración de enunciados que permiten expresar los pensamientos e ideas claramente.

En la programación la sintaxis cuenta con reglas que permitirán expresar lo que nosotros queremos realizar, de manera adecuada y así considere el conjunto de sentencias de nuestro código, como correctas.

* Nota.- Cabe resaltar que la sintaxis de este lenguaje, está inspirado en gran medida a C, Java y Perl.

Y las características de su sintaxis son las siguientes:

No se tienen en cuenta los espacios en blanco y las nuevas líneas: el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor. (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)

var palabra = "Hola";

var saludo =

"Hola Mundo"

Diferencia las letras mayúsculas de las minúsculas, es decir case sensitive. Lo que significa que si definimos estas variables, de la siguiente manera, ambas se crearan sin ningún problema y con su valor asignado.

var numero = 1;

var Numero = 2;

No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.

var dato = 1;

dato = "Tutorial Sintaxis JavaScript";

Se pueden incluir comentarios: los comentarios se utilizan para añadir información en el código fuente del programa. Los comentarios son partes del programa que el intérprete no lee. En JavaScript distinguimos dos formas de comentarios: por línea y por bloque (de varias líneas). En por línea el comentario debe estar antepuesto por dos slash.

//comentario de una sola linea

Y el por bloque el comentario comienza por un slash seguido de un asterisco y termina con un asterisco seguido de un slash.

/\*

\* Este es un comentario

\* de multiple linea

\*/

javascript, el console.log("texto")

es donde se imprime lo que sea

y para crear variables ya no se requiere el tipo de dato

por enjemplo en c++ seria asi:

int numero =10;

char caracter = 'c';

String cadena = "mi texto ";

en javascript seria esto

var numero = 10;

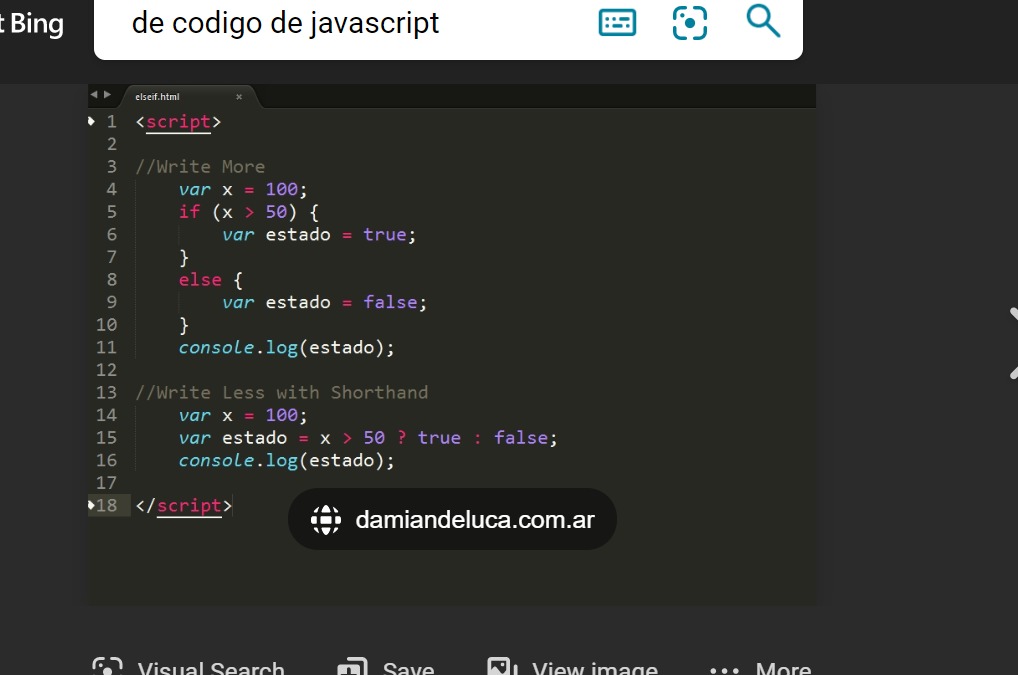
var caracter = 'c';

var cadena = "mi cadena";

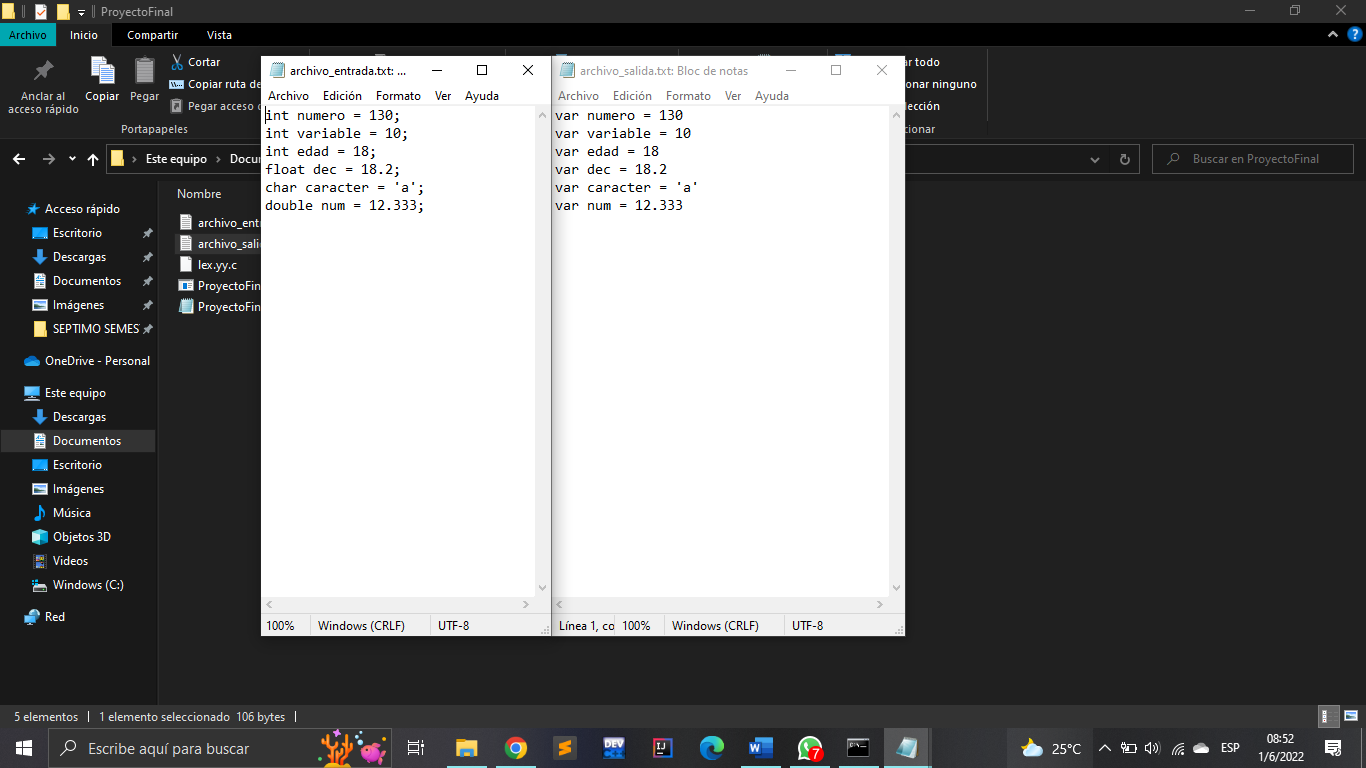
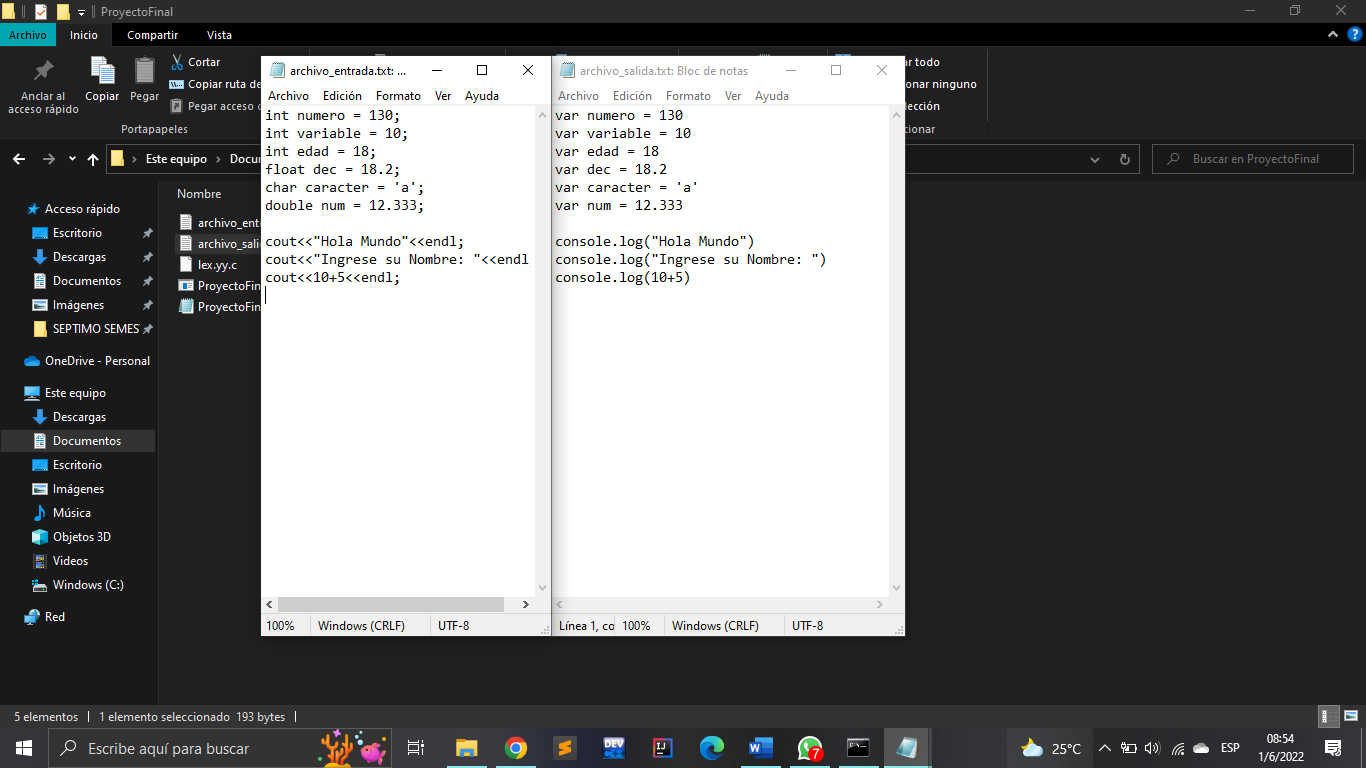
en lugar de var se puede usar let o const

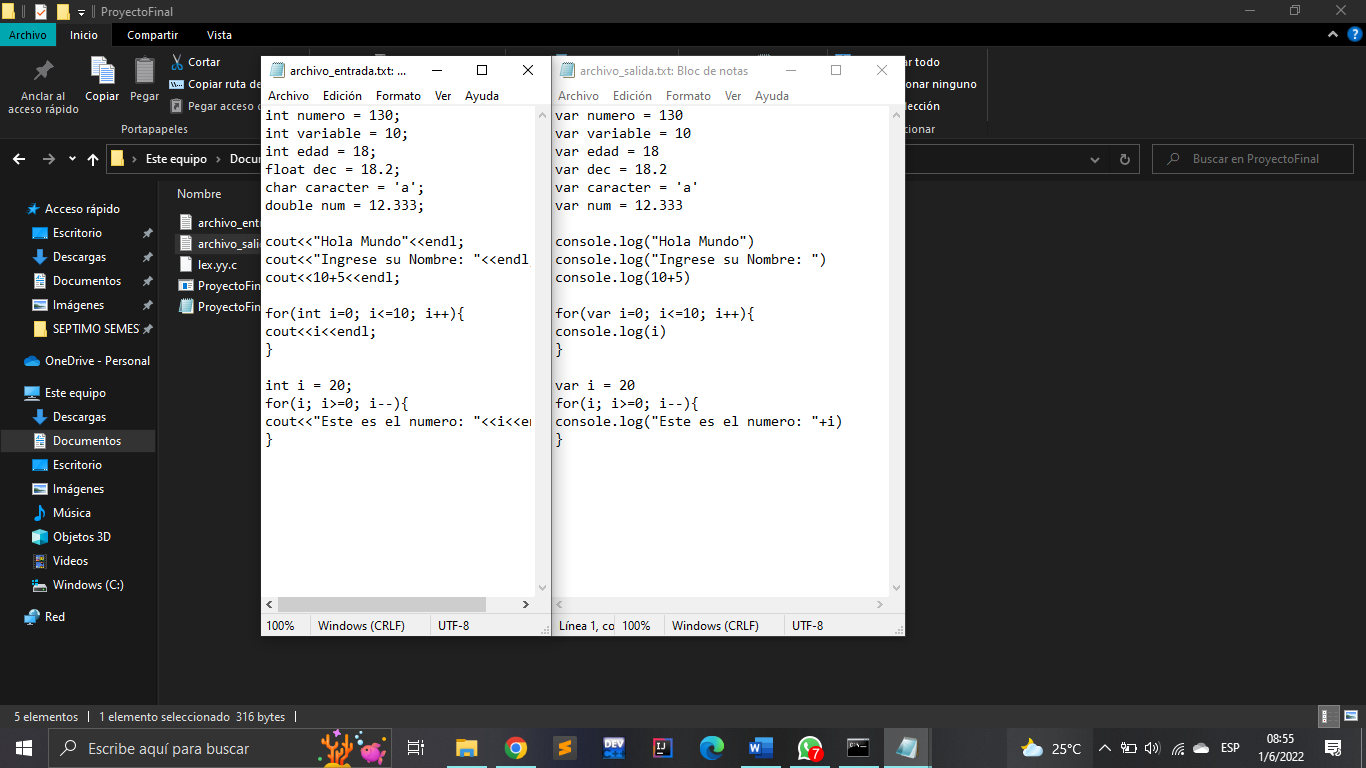
let = que el valor puede cambiar en cualquier momento

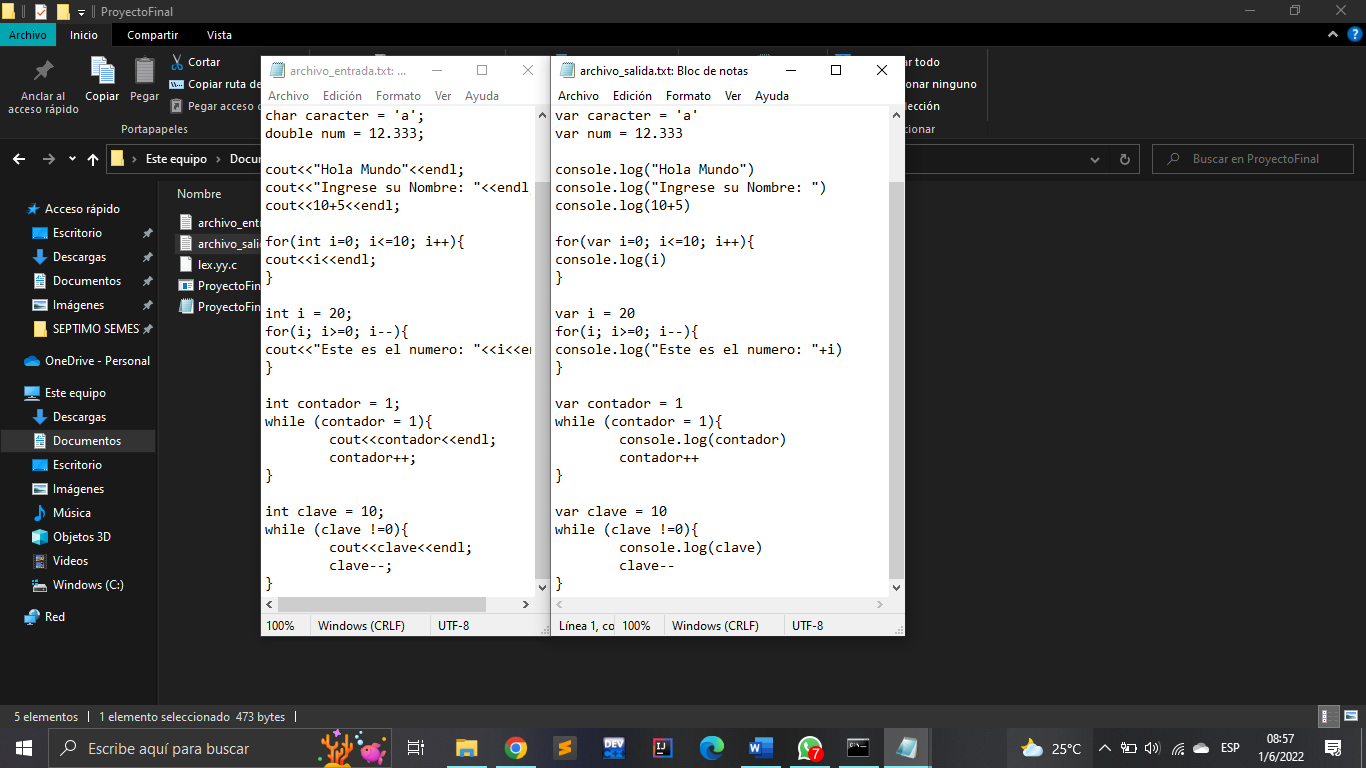
const = el valor nunca cambiara



# Pruebas de software







# Código Fuente

<https://drive.google.com/file/d/1_Su8UllQKG01-qhFjumNaKVHtLVvRvGh/view?usp=sharing>

# Conclusiones

El proyecto finalmente cumplió con parte del cometido final, debido a que se tuvo algunos inconvenientes en el trabajo con bison, su estructura es algo complicada, por lo que se tuvo que hacer una exhaustiva investigación acerca de su funcionamiento y de cómo implementarla en nuestro proyecto final de compiladores.

# Bibliografía

*Blog de compiladores*. (s.f.). Obtenido de https://analizadorsemanticolab1.blogspot.com/2017/08/complidor-con-bison-y-flex.html

*Monografías*. (s.f.). Obtenido de https://www.monografias.com/trabajos58/guia-bison-flex/guia-bison-flex