

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1 A



Manual de Usuario

Wilber Steven Zúñiga Ruano

Carné: 202006629

Guatemala, mayo 2023

Contenido

Introducción	3
Requerimientos	3
Manual de Usuario.....	3
1. Ventana Inicial.....	3
2. Menú archivo.....	4
3. Archivos CST.....	4
4. Analizar Entrada.....	14
5. Reportes	14
5.1. Árbol de Análisis Sintáctico (AST).....	14
5.2. Tabla de Errores	15
5.3. Tabla de Símbolos	15

Introducción

La aplicación EXREGAN permite compilar el lenguaje de programación “tw”, generar la tabla de símbolos resultante, árbol de análisis sintáctico (AST) y reporte de errores sobre el análisis léxico, sintáctico y semántico.

Requerimientos

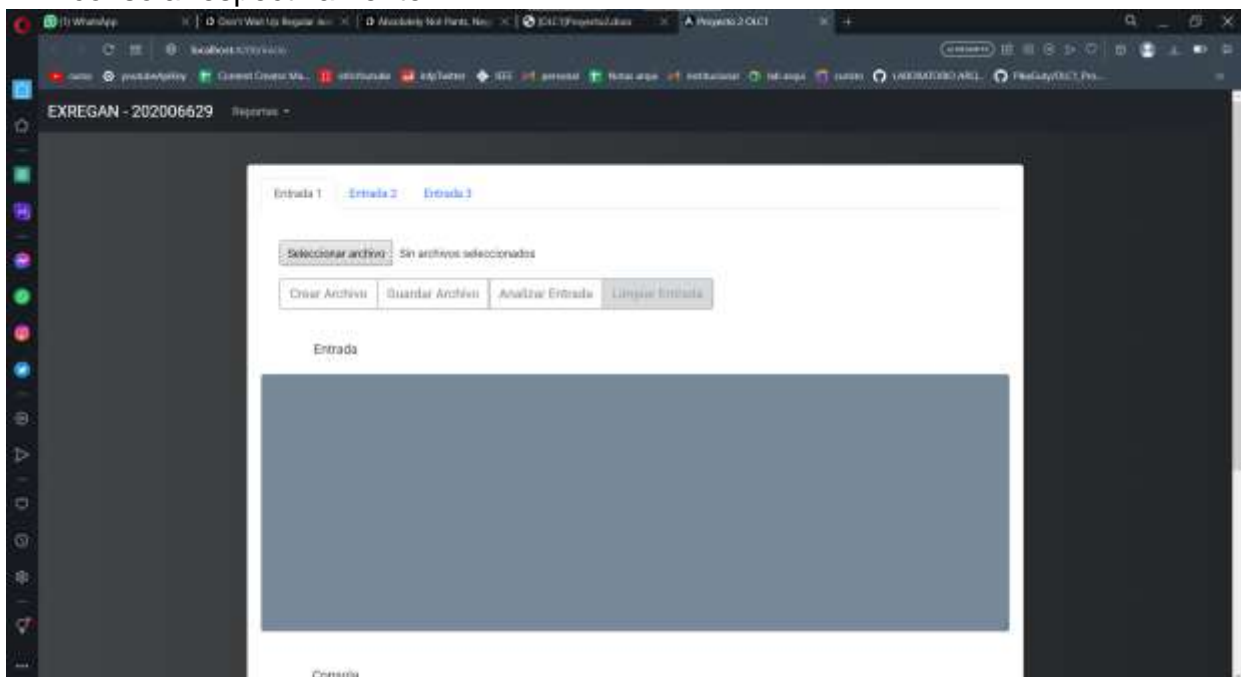
El sistema puede ser utilizado en cualquier sistema operativo que cumpla con los siguientes requerimientos:

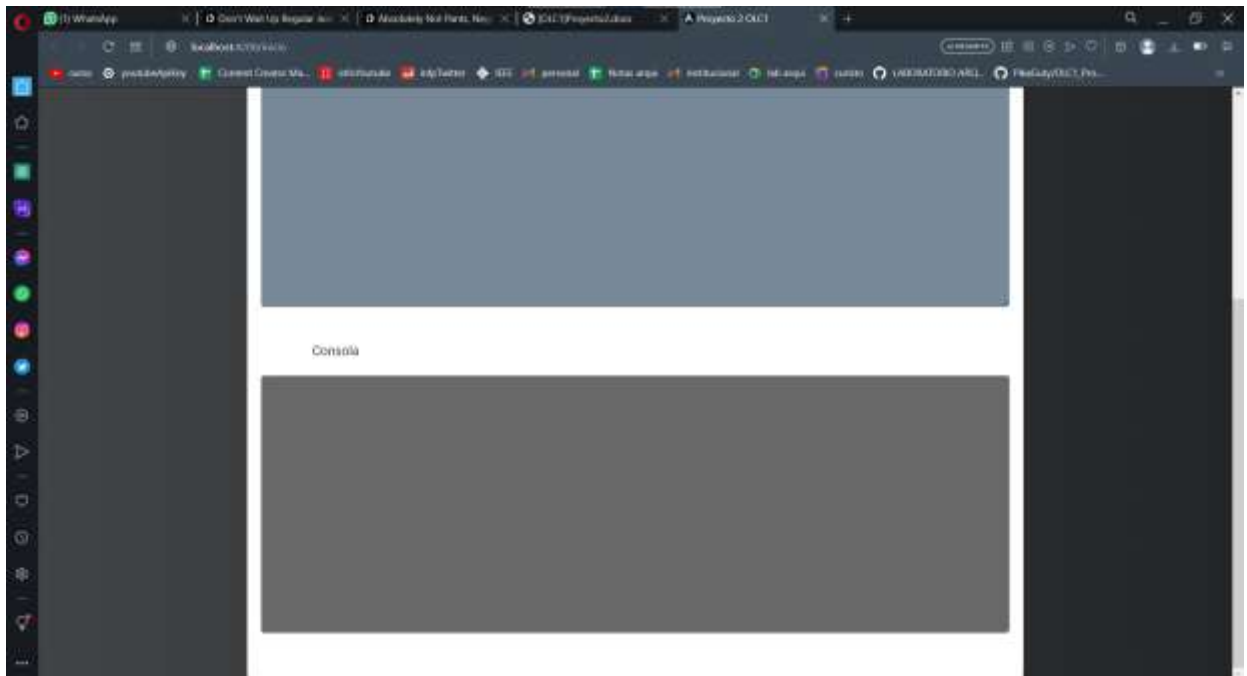
- **Nodejs:** v16.13.2
- **Angular CLI:** 13.1.3

Manual de Usuario

1. Ventana Inicial

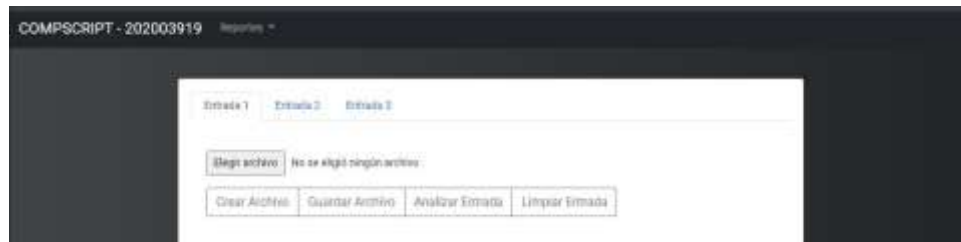
Al iniciar la aplicación se despliega una ventana la cual posee tres ventanas que poseen el menú de archivo y dos áreas de texto, siendo el editor de código y la consola respectivamente.





2. Menú archivo

- **Crear Archivo:** Permite crear un archivo en blanco.
- **Abrir Archivo:** Permite abrir un archivo ya existente. La extensión aceptada por la aplicación es .tw.
- **Guardar Archivo:** Permite guardar el archivo que este en el área de texto.
- **Limpiar Entrada:** Permite vaciar el área de entrada y la consola.



3. Archivos tw

El archivo tw posee el código que se desea ejecutar. La sintaxis que debetener debe ser la siguiente:

- **Comentarios:** Existen dos formas de colocar comentarios, siendo comentarios de una línea y comentarios multilínea.
 - **Comentarios de una línea:** Estos comentarios deberán comenzar con // y terminar con un salto de línea.
 - **Comentarios multilínea:** Estos comentarios deberán comenzar con /* y terminar con */.
- **Tipos de dato:** El lenguaje acepta los siguientes tipos de dato:

TIPO	DEFINICION	DESCRIPCION	EJEMPLO	OBSERVACIONES	DEFAULT
Entero	Int	Este tipo de datos aceptará solamente números enteros.	1, 50, 100, 25552, etc.	Del -2147483648 al 2147483647	0
Doble	Double	Admite valores numéricos con decimales.	1.2, 50.23, 00.34, etc.	Se manejará cualquier cantidad de decimales	0.0
Booleano	Boolean	Admite valores que indican verdadero o falso.	True, false	Si se asigna un valor booleano a un entero se tomará como 1 o 0 respectivamente.	True
Caracter	Char	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples. "	'a', 'b', 'c', 'E', 'Z', '1', '2', '!', '%', ' ', '!', '=', '!', '&', '!', '\\', 'n', etc.	En el caso de querer escribir comilla simple se escribirá \' y después comilla simple \', si se quiere escribir \\ se escribirá dos veces \\. Existirá también \n, \t, \r, \".	'\u0000' (carácter 0)
Cadena	String	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles, "	"cadena1", "...", "cadena 1"	Se permitirá cualquier carácter entre las comillas dobles, incluyendo las secuencias de escape: \", comilla doble \\ barra invertida \n salto de línea \r retorno de carro \t tabulación	"" (string vacío)

- **Secuencias de Escape:** Son sentencias que pueden ser parte de las cadenas que representan caracteres especiales

SECUENCIA	DESCRIPCION	EJEMPLO
\n	Salto de línea	"Hola\nMundo"
\\	Barra invertida	"C:\\miCarpeta\\Personal"
\"	Comilla doble	"\"Esto es una cadena\""
\t	Tabulación	"\tEsto es una tabulación"
\'	Comilla Simple	"\'Estas son comillas simples\'"

- **Operadores Aritméticos:** Las operaciones aritméticas que se pueden realizar son las siguientes:

- **Suma:** Se realiza la suma de dos expresiones utilizando el signo +. Los casos válidos para esta son los siguientes:

+	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble	Entero	Entero	Cadena
Doble	Doble	Doble	Doble	Doble	Cadena
Boolean	Entero	Doble			Cadena
Caracter	Entero	Doble		Cadena	Cadena
Cadena	Cadena	Cadena	Cadena	Cadena	Cadena

- **Resta:** Se realiza la resta de dos expresiones utilizando el signo -. Los casos válidos para esta son los siguientes:

-	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble	Entero	Entero	
Doble	Doble	Doble	Doble	Doble	
Boolean	Entero	Doble			
Caracter	Entero	Doble			
Cadena					

- **Multiplicación:** Se realiza la multiplicación de dos expresiones utilizando el signo *. Los casos válidos para esta son los siguientes:

*	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble		Entero	
Doble	Doble	Doble		Doble	
Boolean					
Caracter	Entero	Doble			
Cadena					

- **División:** Se realiza la división de dos expresiones utilizando el signo /. Los casos válidos para esta son los siguientes:

/	Entero	Doble	Boolean	Caracter	Cadena
Entero	Doble	Doble		Doble	
Doble	Doble	Doble		Doble	
Boolean					
Caracter	Doble	Doble			
Cadena					

- **Potencia:** Se realiza la potencia de una expresión elevada a otra utilizando el signo \wedge . Los casos válidos para esta son los siguientes:

\wedge	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble			
Doble	Doble	Doble			
Boolean					
Caracter					
Cadena					

- **Modulo:** Se realiza la operación de resto de división de un numero entre otro utilizando el signo $\%$. Los casos validos para esta son los siguientes:

$\%$	Entero	Doble	Boolean	Caracter	Cadena
Entero	Doble	Doble			
Doble	Doble	Doble			
Boolean					
Carácter					
Cadena					

- **Negación Unaria:** Niega el valor de una expresión utilizando el signo $-$ antes de esta. Los casos validos para esta son los siguientes:

$-$ número	Resultado
Entero	Entero
Doble	Doble
Boolean	
Carácter	
Cadena	

- **Operadores Relacionales:** Son las operaciones que permiten comparar el valor de dos expresiones, como las siguientes:
 - **Igualación:** Comparación entre valores y verifica si son iguales. Si estos valores son iguales retorna true, caso contrario retorna false. Esta se realiza con el operador $==$.
 - **Diferenciación:** Comparación entre valores y verifica si son iguales. Si estos valores son iguales retorna false, caso contrario retorna true. Esta se realiza con el operador $!=$.
 - **Menor que:** Compara ambos lados y verifica si el derecho es mayor que el izquierdo. Si esto se cumple retorna true, caso contrario retorna false. Esta se realiza con el operador $<$.

- **Menor o igual que:** Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo. Si esto se cumple retorna true, caso contrario retorna false. Esta se realiza con el operador <=.
- **Mayor que:** Compara ambos lados y verifica si el izquierdo es mayor que el derecho. Si esto se cumple retorna true, caso contrario retorna false. Esta se realiza con el operador >.
- **Mayor o igual que:** Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho. Si esto se cumple retorna true, caso contrario retorna false. Esta se realiza con el operador >=.
- **Operadores lógicos:** Son los encargados de realizar las comparaciones lógicas
 - **OR:** Compara expresiones lógicas y si al menos una es verdadera entonces devuelve verdadero en otro caso retorna falso. Esta se realiza con el operador ||.
 - **AND:** Compara expresiones lógicas y si ambas son verdaderas entonces retorna verdadero en otro caso retorna falso. Esta se realiza con el operador &&.
 - **NOT:** Devuelve el valor inverso de una expresión lógica si esta es verdadera retorna falso, de lo contrario retorna verdadero. Esta se realiza con el operador !.
- **Signos de Agrupación:** Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por (y).
- **Declaración y Asignación de Variables:** Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador.
 Durante la declaración de variables también se tendrá la opción de poder crear múltiples variables al mismo tiempo, al crear múltiples variables al mismo tiempo se tendrá la opción de crear todas las variables con un mismo valor, para ello se realizará una asignación al final del listado de las variables, en caso de no indicar esta asignación se dejará el valor por defecto para cada variable.
 Las variables no pueden cambiar de tipo de dato, se deben mantener con el tipo declarado inicialmente, por lo que se debe de validar que el tipo de la variable y el valor sean compatibles

```
<TIPO> identificador;  
<TIPO> identificador = <EXPRESION>;
```


- **Casteos:** Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo. Para hacer esto, se colocará la palabra reservada del tipo de dato destino entre paréntesis seguido de una expresión. El lenguaje aceptará los siguientes casteos:

- Int a Double
- Double a Int
- Int a String
- Int a Char
- Double a String
- Char a Int
- Char a Double

```
'(' <TIPO> ')' <EXPRESION>
```

- **Incremento y Decremento:** Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de un valor de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria.

```
<EXPRESION> '+' '+' ';'
<EXPRESION> '-' '-' ';'

```

- **Vectores:** Los vectores son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string. El lenguaje permitirá únicamente el uso de arreglos de una o dos dimensiones.

- **Declaración:**

```
//DECLARACION TIPO 1
<TIPO> '[' ']' <ID> = new <TIPO> '[' <EXPRESION> ']' ';'

//DECLARACION TIPO 2
<TIPO> '[' ']' <ID> = '{' <LISTAVALORES> '}' ';'

//Ejemplo de declaración tipo 1
int[ ] vector1 = new int[4]; //se crea un vector de 4 posiciones, con 0 en cada posición

//Ejemplo de declaración tipo 2
string[ ] vector2 = {"hola", "Mundo"}; //vector de 2 posiciones, con "Hola" y "Mundo"
```

- **Acceso**

```
<ID> '[' EXPRESION ']'
```

- **Modificación:**

```
<ID> '[' EXPRESION ']' = EXPRESION ';'

```

- **Sentencias de Control**

- **IF:** La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia.
- **IF ELSE:** La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia y se ejecutan las instrucciones del else.
- **ELSE IF:** La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia y se hace la verificación en el siguiente else if.

```

'if' '(' [<EXPRESION>] ')' '{
    [<INSTRUCCIONES>]
}'
| 'if' '(' [<EXPRESION>] ')' '{
    [<INSTRUCCIONES>]
}' 'else' '{
    [<INSTRUCCIONES>]
}'
| 'if' '(' [<EXPRESION>] ')' '{
    [<INSTRUCCIONES>]
}' 'else' [<IF>]

```

- **Operador Ternario:** El operador ternario es un operador que hace uso de 3 operandos para simplificar la instrucción 'if' por lo que a menudo este operador se le considera como un atajo para la instrucción 'if'. El primer operando del operador ternario corresponde a la condición que debe de cumplir una expresión para que el operador retorne como valor el resultado de la expresión segundo operando del operador y en caso de no cumplir con la expresión el operador debe de retornar el valor de la expresión del tercer operando del operador.

```

<CONDICION> '?' <EXPRESION> ':' <EXPRESION>

```

- **Switch Case:** Es una estructura utilizada para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos if.

- **Switch:** Estructura principal del switch, donde se indica la expresión a evaluar.

```
'switch' '(' [<EXPRESION> ] ')' '{
    [<CASES_LIST>] [<DEFAULT>]
}'
| 'switch' '(' <EXPRESION> ')' '{
    [<CASES_LIST>]
}'
| 'switch' '(' <EXPRESION> ')' '{
    [<DEFAULT>]
}'
```

- **Case:** Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch.

```
'case' [<EXPRESION>] ':'
    [<INSTRUCCIONES>]
```

- **Default:** Estructura que contiene las sentencias si en dado caso no haya salido del switch por medio de una sentencia break.

```
'default' ':'
    [<INSTRUCCIONES>]
```

- **Sentencias Ciclicas:** Son un conjunto de instrucciones que se realizan siempre y cuando la condición sea verdadera.

- **While:**

```
'while' '(' [<EXPRESION> ] ')' '{
    [<INSTRUCCIONES>]
}'
```

- **For:**

```
'for' '(' ([<DECLARACION>|<ASIGNACION>]); [<CONDICION>]; [<ACTUALIZACION> ] ')' '{
    [<INSTRUCCIONES>]
}'
```

- **Do-While:**

```
'do' '{
    [<INSTRUCCIONES>]
}' 'while' '(' [<EXPRESION> ] ':'
```

- **Sentencias de Transferencia:** Las sentencias de transferencia nos permiten manipular el comportamiento de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias:

- **Break:** Hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo.

break;

- **Continue:** Puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error.

continue;

- **Return:** Finaliza la ejecución de un método o función y puede especificar un valor para ser devuelto a quien llama a la función.

```
return;
return <EXPRESION>;
```

- **Funciones:** Las funciones serán declaradas definiendo primero su identificador único, luego un tipo de dato para la función, seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no utilice parámetros). Cada parámetro debe estar compuesto por su tipo seguido de un identificador, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis. Luego de definir la función y sus parámetros se declara el cuerpo de la función, el cual es un conjunto de instrucciones delimitadas por llaves {}. Para las funciones es obligatorio que las mismas posean un valor de retorno que coincida con el tipo con el que se declaró la función. **NO HAY SOBRECARGA DE METODOS.**

```
<TIPO> <ID> '(' [<PARAMETROS>] ')' '{'
    [<INSTRUCCIONES>
    '}'
PARAMETROS -> [<PARAMETROS> ',' [<TIPO>] [<ID>]
              | [<TIPO>] [<ID>]
```

- **Métodos:** los métodos serán declarados haciendo uso un identificador del método, seguido de la palabra reservada 'void' (que puede o no aparecer), seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no

utilice parámetros). Cada parámetro debe estar compuesto por su tipo seguido de un identificador, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis. Luego de definir el método y sus parámetros se declara el cuerpo del método, el cual es un conjunto de instrucciones delimitadas por llaves {}.

NO HAY SOBRECARGA DE METODOS.

```
'void' <ID> '(' [<PARAMETROS>] ')' '{  
    [<INSTRUCCIONES>  
}'  
PARAMETROS -> [<PARAMETROS>] ',' [<TIPO>] [<ID>]  
            | [<TIPO>] [<ID>]
```

- **Llamadas:** La llamada a una función especifica la relación entre los parámetros reales y los formales y ejecuta la función. La llamada a una función devuelve un resultado que ha de ser recogido, bien asignándolo a una variable del tipo adecuado, bien integrándolo en una expresión. La sintaxis de las llamadas de los métodos y funciones serán la misma.

```
LLAMADA -> [<ID>] '(' [<PARAMETROS_LLAMADA>] )'  
          | [<ID>] '(' '  
PARAMETROS_LLAMADA -> [<PARAMETROS_LLAMADA>] ',' [<EXPRESION>]  
                    | [<EXPRESION>]
```

- **Función Print:** Función que nos permite imprimir expresiones con valores con valores únicamente de tipo entero, double, booleano, cadena y carácter. no concatena un salto de línea al final del contenido que recibe como parámetro.

```
'print' '(' <EXPRESION> ')';
```

- **Función toLower:** Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas

```
'toLower' '(' <EXPRESION> ')';
```

- **Función toUpper:** Esta función recibe como parámetro una expresión de tipo cadena retorna una nueva cadena con todas las letras mayúsculas.

```
'toUpper' '(' <EXPRESION> ')';
```

- **Round:** Esta función recibe como parámetro un valor numérico. Permite redondear los números decimales según las siguientes reglas:
 - Si el decimal es mayor o igual que 0.5, se aproxima al número superior
 - Si el decimal es menor que 0.5, se aproxima al número inferior.

```
'round' '(' <VALOR> ')';
```

- **Funciones Nativas:**

- **Length:** Esta función recibe como parámetro un vector, una lista o una cadena y devuelve el tamaño de este.

```
'length' '(' <VALOR> ')';
```

- **Typeof:** Esta función retorna una cadena con el nombre del tipo de dato evaluado.

```
'typeof' '(' <VALOR> ')';
```

- **ToString:** Esta función permite convertir un valor de tipo numérico o booleano en texto.

```
'toString' '(' <VALOR> ')';
```

- **toCharArray:** Esta función permite convertir una cadena en un vector de caracteres.

```
'toArray' '(' <VALOR> ')';
```

- **Main:** Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia MAIN para indicar que método o función es la que iniciará con la lógica del programa.

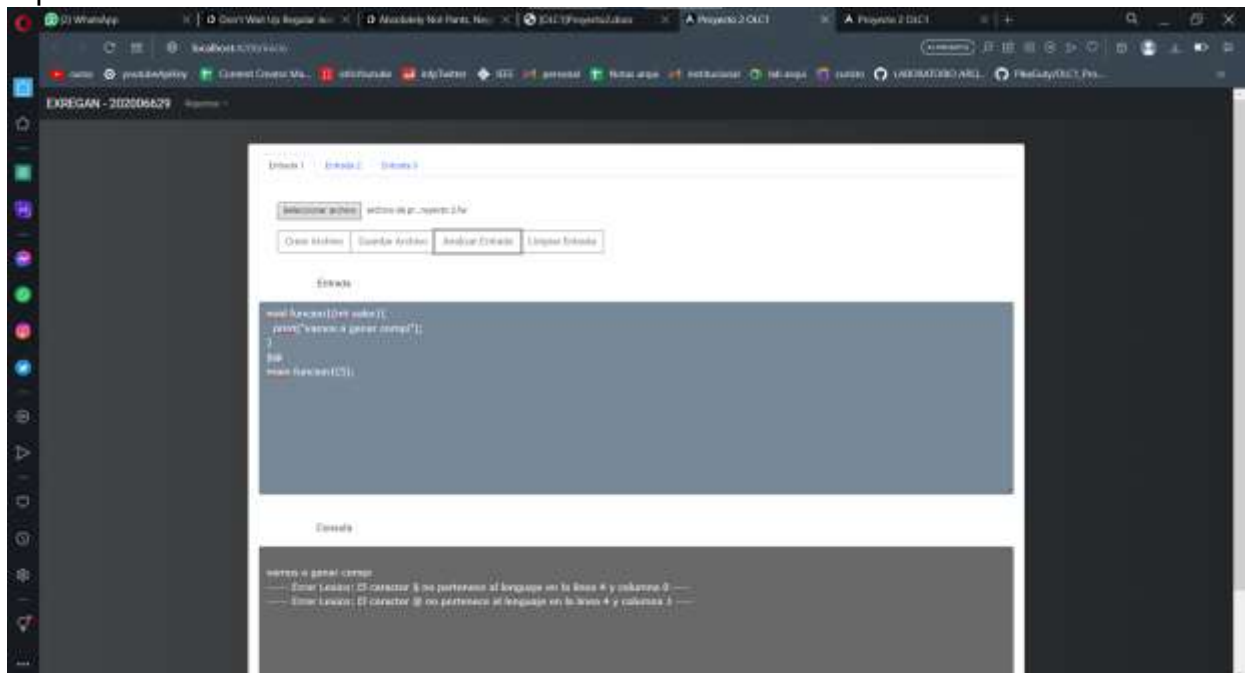

```

'main' <ID> '(' ')' ';'
'main' <ID> '(' <LISTAVALORES> ')' ';'
LISTAVALORES->LISTAVALORES ',' EXPRESION
| EXPRESION

```

4. Analizar Entrada

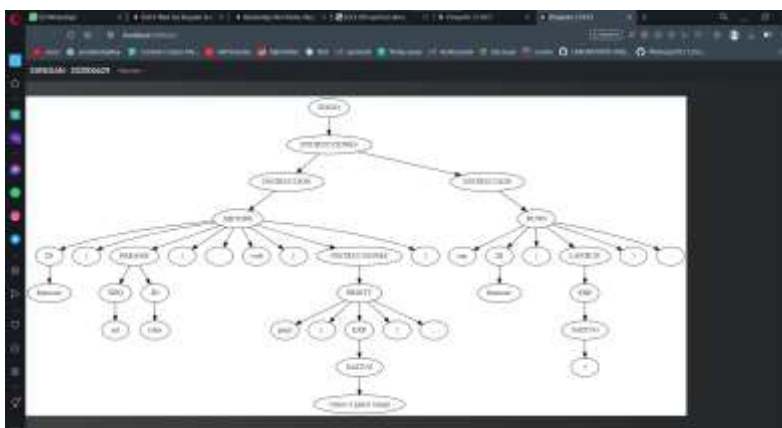
Realiza el análisis del código que se encuentre en el área de entrada y coloca las respectivas salidas en la consola. Al finalizar el análisis se podrán ver los reportes.



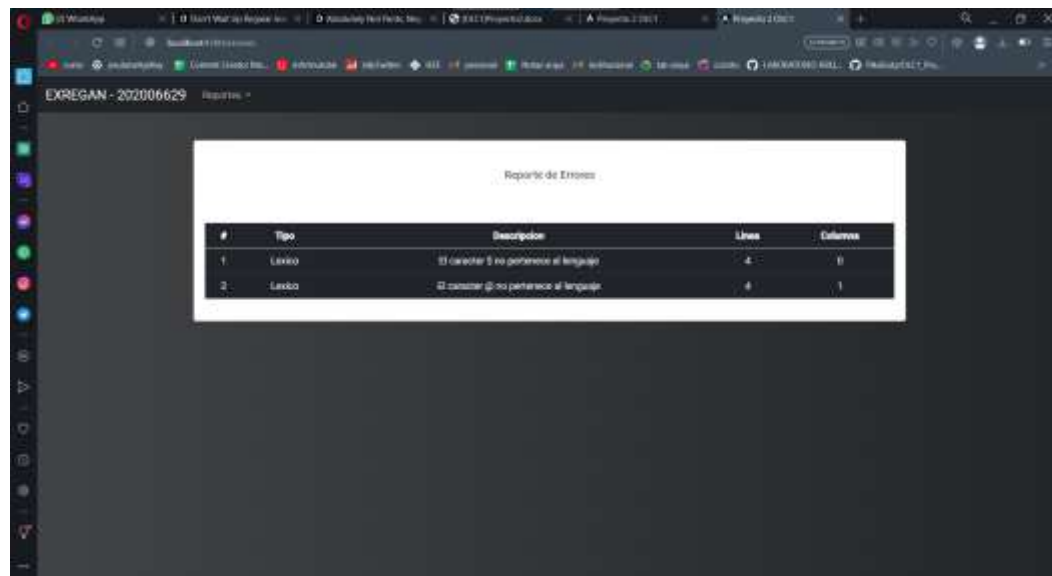
5. Reportes

Al finalizar el análisis del código o archivo del área de entrada será posible ver los reportes de este, siendo estos los siguientes:

5.1. Árbol de Análisis Sintáctico (AST)



5.2. Tabla de Errores



Reporte de Errores

#	Tipo	Descripción	Línea	Columna
1	Lexico	El carácter S no pertenece al lenguaje	4	0
2	Lexico	El carácter @ no pertenece al lenguaje	4	1

5.3. Tabla de Símbolos

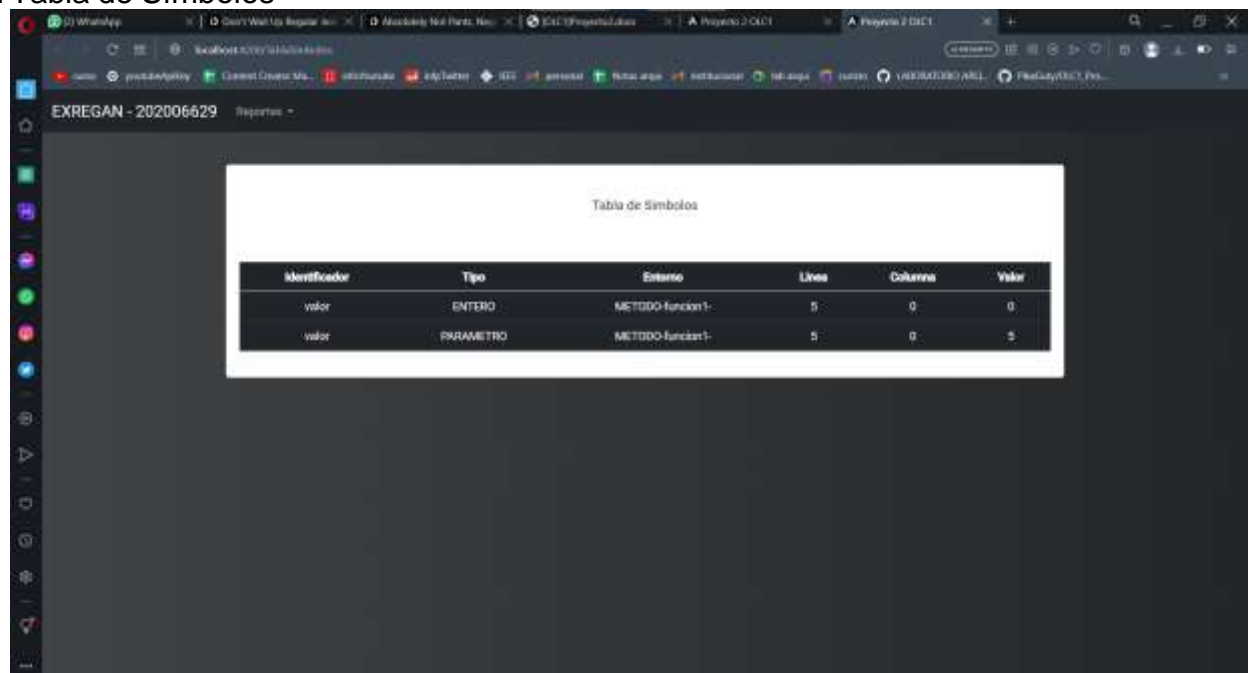


Tabla de Símbolos

Identificador	Tipo	Entorno	Línea	Columna	Valor
valor	ENTERO	METODO-funcion1-	5	0	0
valor	PARAMETRO	METODO-funcion1-	5	0	5