

</> MFM SCRIPT

PROYECTO 2 - COMPILADORES 1



MANUAL DE USUARIO

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Rodrigo Alejandro Hernández de León

201900042

I. Introducción

1. Objetivo

Otorgar al usuario un apoyo documentado del uso de la interfaz gráfica al igual del manejo del código y poder obtener una ejecución exitosa de su entrada de texto, sin errores y con éxito uso de la herramienta.

2. Requerimientos

- Computadora portátil o de escritorio.
- Mínimo 1GB de Memoria RAM
- Cualquier sistema operativo que contenga un navegador web.
- Graphviz
- Node JS
- React
- Resolución gráfica máximo 1900 x 1070 píxeles.

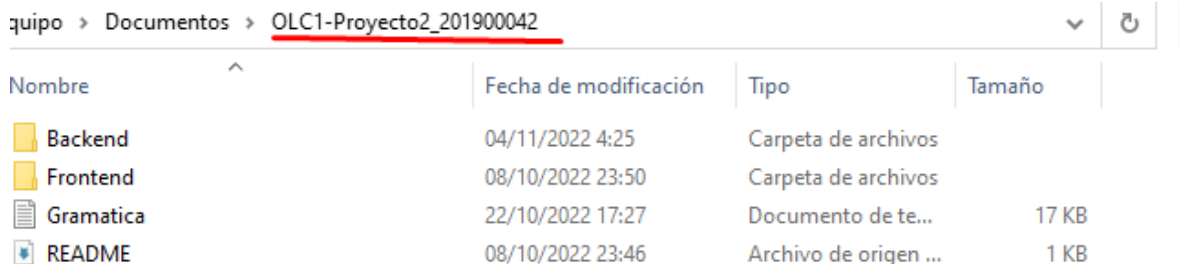
II. Opciones del Sistema





En las siguientes páginas se estará explicando el uso de cada una de las funcionalidades que tiene la aplicación:

1. Ingreso a la aplicación.
2. Lenguaje OLC.
3. Ejecución de código
4. Archivo:
 - i. Crear Archivo
 - ii. Abrir Archivo.
 - iii. Guardar Como.
5. Reporte:
 - i. Tabla de Simbolos
 - ii. Errores.
 - iii. AST.
6. Ver

1. Ingreso a la aplicación

Para poder ingresar a la aplicación es necesario ubicarse en la carpeta del proyecto y encontrar la siguiente aplicación ejecutable:



quipo > Documentos > <u>OLC1-Proyecto2_201900042</u>				
Nombre	Fecha de modificación	Tipo	Tamaño	
 Backend	04/11/2022 4:25	Carpeta de archivos		
 Frontend	08/10/2022 23:50	Carpeta de archivos		
 Gramatica	22/10/2022 17:27	Documento de te...	17 KB	
 README	08/10/2022 23:46	Archivo de origen ...	1 KB	

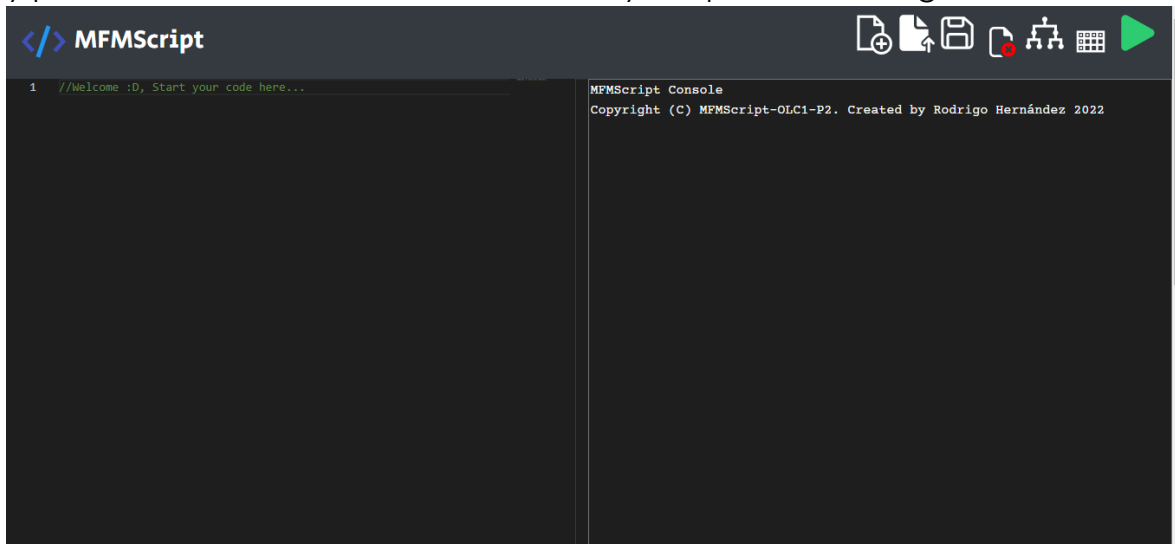
Posteriormente abrir la carpeta de Backend, abrir el CMD y ejecuta el siguiente comando:

```
npm start
```

Asimismo abre la carpeta del Frontend y dentro abrir la carpeta de mfmscript y posteriormente abrir el CMD para ejecutar el siguiente comando:

```
npm run dev
```

y posteriormente abra el localhost:3000 y le aparecera la siguiente ventana:



2. Lenguaje OLC

En esta parte se encuentra la descripción del manejo del lenguaje OLC.

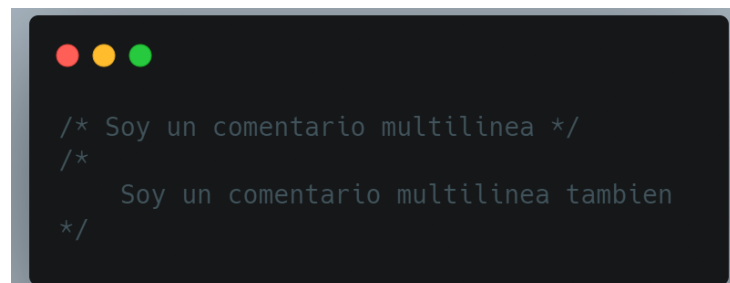
a. Comentarios

Esta instrucción puede estar en cualquier parte del código. Existen dos tipo de comentarios:

- Comentario de una línea: iniciaran con los caracteres "//", el cual indicará que en adelante todos los caracteres serán considerados como comentarios y el fin del comentario estará delimitado por un salto de línea.



- Comentario multilinea: estarán encerrados dentro de los caracteres "/*" y "*/".



b. Variables (Identificadores) y nombre de Funciones y Métodos

Para declarar o llamar un nombre de una variable, nombrar u ejecutar una función o método es obligatorio tener una letra al inicio y poder contener un numero o guion bajo para ser aceptado.

c. Tipo de Datos:

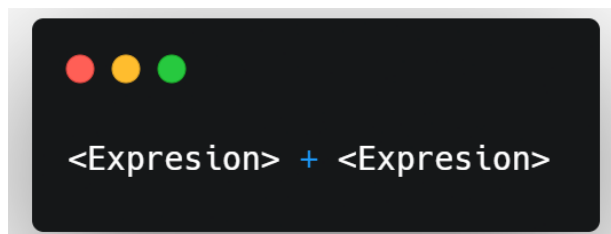
Palabra Reservada	Descripción	Ejemplos
int	Conjunto de solo dígitos.	4568 5 94
double	Conjunto de dígitos que tienen una única vez el carácter punto dentro del conjunto de caracteres.	12.5 45323.7895 3.1416
string	Conjunto de caracteres dentro de comillas dobles.	"Esta es una cadena" "hola mundo" "organización de lenguajes y compiladores" 1"
boolean	Tipo de dato verdadero o falso.	true false
char	Un carácter dentro de una comilla simple.	'o' 'L' '5'

d. Operadores

En todas las operaciones pueden venir 2 o más operandos.

i. Operadores Aritméticos

Suma:



Resta

```
<Expresion> - <Expresion>
```

Multiplicación

```
<Expresion> * <Expresion>
```

División

```
<Expresion> / <Expresion>
```

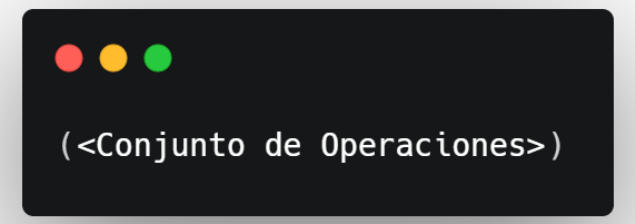
Potencia

```
<EXPRESION> ^ <EXPRESION>
```

Módulo

```
<EXPRESION> % <EXPRESION>
```

Paréntesis



```
(<Conjunto de Operaciones>)
```

i. Operadores Relacionales

Mayor



```
<EXPRESSION> > <EXPRESSION>
```

Menor



```
<EXPRESSION> < <EXPRESSION>
```

Mayor o igual que



```
<EXPRESSION> >= <EXPRESSION>
```

Menor o igual que

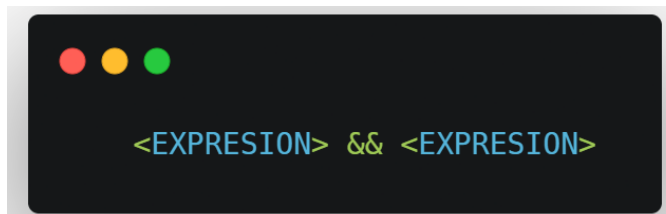
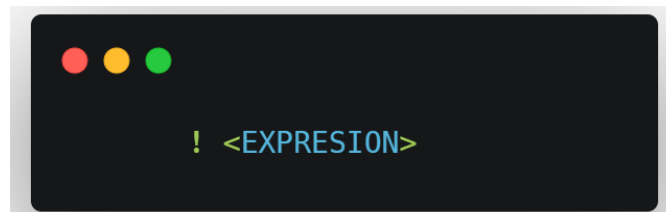


```
<EXPRESSION> <= <EXPRESSION>
```

Igual



```
<EXPRESSION> == <EXPRESSION>
```

Diferente**ii. Operadores Lógicos****Or****And****Not****e. Declaración**

Esta instrucción permite ingresar variables dentro del flujo del código. Para ingresar el nombre se necesita reconocer la palabra reservada "ingresar" y el tipo de dato. A continuación, un ejemplo de una declaración:


```
//DECLARACION ENTEROS
int a;
int b = 0;
//DECLARACION DECIMALES
double c;
double d = 4.5;
//DECLARACION CARACTERES
char e;
char f = 'a';
//DECLARACION BOOLEANOS
boolean g;
boolean h = true;
boolean i = false;
//DECLARACION CADENAS
string j;
string k = "hola mundo";
```

Es posible realizar declaraciones múltiples, para ello es necesario ingresar una lista de nombres, separadas por comas. A continuación, un ejemplo de una declaración múltiple:

```
int a,b,c,d = 10;
```

f. Asignación

Esta instrucción cambia el valor de una variable determinada y la actualiza con el valor de una expresión, su estructura consta del nombre de la variable y una expresión. A continuación, un ejemplo de una asignación:

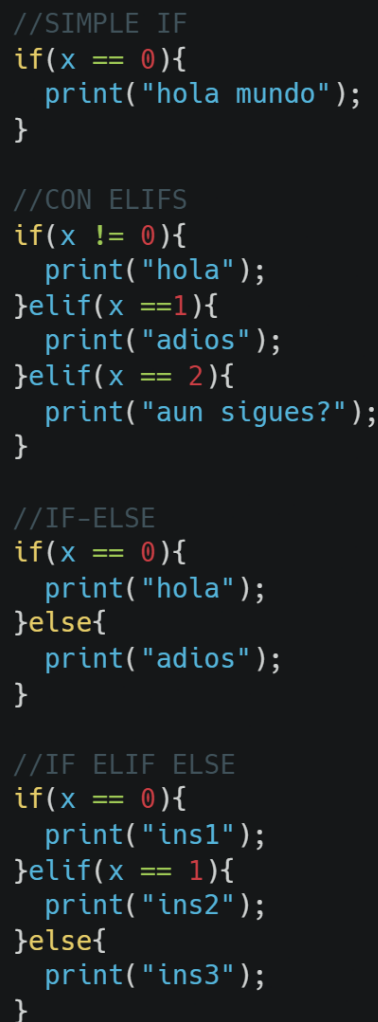
```
a = 20;
a++;
```

g. Condicional If

Esta es una instrucción que permite ejecutar un bloque de instrucciones cuando una condición es válida. La instrucción necesita una condición para

ejecutar un bloque asignado, cuando la condición es falsa se ejecuta otro bloque de instrucciones. Es posible que el bloque de instrucciones que ejecuta una condición falsa, sea opcional. Es obligatorio que ingrese una condición en esta instrucción.

Caso especial: se usará la palabra reservada "elif" cuando una condición es falsa y se trata de volver hacer otra validación con otra condición. Este tipo de ejecución puede ser opcional o repetirse una o muchas veces. A continuación, se muestra un ejemplo de este caso especial.



```
//SIMPLE IF
if(x == 0){
    print("hola mundo");
}

//CON ELIFS
if(x != 0){
    print("hola");
}elif(x ==1){
    print("adios");
}elif(x == 2){
    print("aun sigues?");
}

//IF-ELSE
if(x == 0){
    print("hola");
}else{
    print("adios");
}

//IF ELIF ELSE
if(x == 0){
    print("ins1");
}elif(x == 1){
    print("ins2");
}else{
    print("ins3");
}
```

h. Switch case

Este tipo de condición permite ejecutar una lista de instrucciones en base a un valor ingresado, existen varias opciones posibles y cuando el valor coincida con una de las opciones se ejecuta un conjunto de instrucciones. Existe una palabra reservada "default" para ejecutar automáticamente cuando la lista de opciones no se cumple, pero es de forma opcional.

```
switch(x){  
  case 1:  
    print("hi");  
    break;  
  case 2:  
    print("halo");  
    break;  
  case 3:  
    print("hello");  
    break;  
  default:  
    print("hola");  
    break;  
}
```


i. Ciclo For

Este ciclo ejecuta un conjunto de instrucciones, con un límite de repeticiones. Es necesario ingresar un valor inicial y también un valor final, el valor final es el que le indica al ciclo, en momento para terminar de realizar las repeticiones. Otro de los elementos necesarios en este ciclo es el número de pasos que realizará entre cada repetición. Cuando el ciclo no tiene definido el número de pasos, se tomará como defecto el incremento en 1. Es posible que la lista de instrucciones esté vacía.

```
for(int i = 0; i < 5; i++){  
  println(i);  
}  
for(i = 10; i > 0; i--){  
  println(i);  
}
```

j. Ciclo while

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. Para poder realizar una repetición es necesario que exista una condición. Es posible que la lista de instrucciones esté vacía.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains the following code:

```
while(x!=10){  
    println(x);  
    x++;  
}
```

k. Ciclo do while

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. A diferencia del ciclo anterior, este ciclo realiza una única repetición sin restricción. Para poder realizar las demás repeticiones es necesario que exista una condición. Es posible que la lista de instrucciones esté vacía.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains the following code:

```
do{  
    print("hola");  
    x++;  
}while(x != 10);
```

l. Ciclo do until

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. A diferencia del ciclo anterior, este ciclo realiza una única repetición sin restricción. Para poder parar las repeticiones necesita de una condición. Es posible que la lista de instrucciones esté vacía.

```
do{
  print("hola");
  x++
}until(x == 10);
```

m. Método

Esta instrucción permite agrupar un conjunto de instrucciones y asignarles un nombre para identificarlo dentro del contenido del archivo. No necesita una instrucción de "Retorno" y si en caso es reconocido este tipo de instrucción, debe reportar dicho error.

```
metodo1():void{
  print("hola desde mi metodo");
}

metodo2(){
  print("holis");
}
```

Es posible agregar parámetros al método, estos parámetros tendrán definido el tipo de dato y su respectivo nombre. Cada uno de los parámetros estará separado por un carácter coma.

```
edad(int x):void{
  print("mi edad es " + x);
}
```

n. Función

Esta instrucción permite agrupar un conjunto de instrucciones y asignarles un nombre para identificarlo dentro del contenido del archivo. Esta instrucción si es posible reconocer una instrucción de "Retorno" en su estructura.

```
suma(): int{  
    int x = 1;  
    int y = 2;  
    return x + y;  
}
```

Es posible agregar parámetros a la función, estos parámetros tendrán definido el tipo de dato y su respectivo nombre. Cada uno de los parámetros estará separado por un carácter coma.

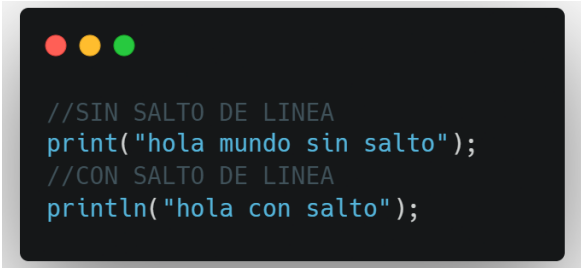
o. Llamada de Funciones y Métodos

Este tipo de instrucción realiza la ejecución de un método o función, para poder realizarlo es necesario ingresar el identificador de la función o método y la lista de parámetros necesarios.

```
suma(int x, int y): int{  
    return x + y;  
}
```

p. Impresión

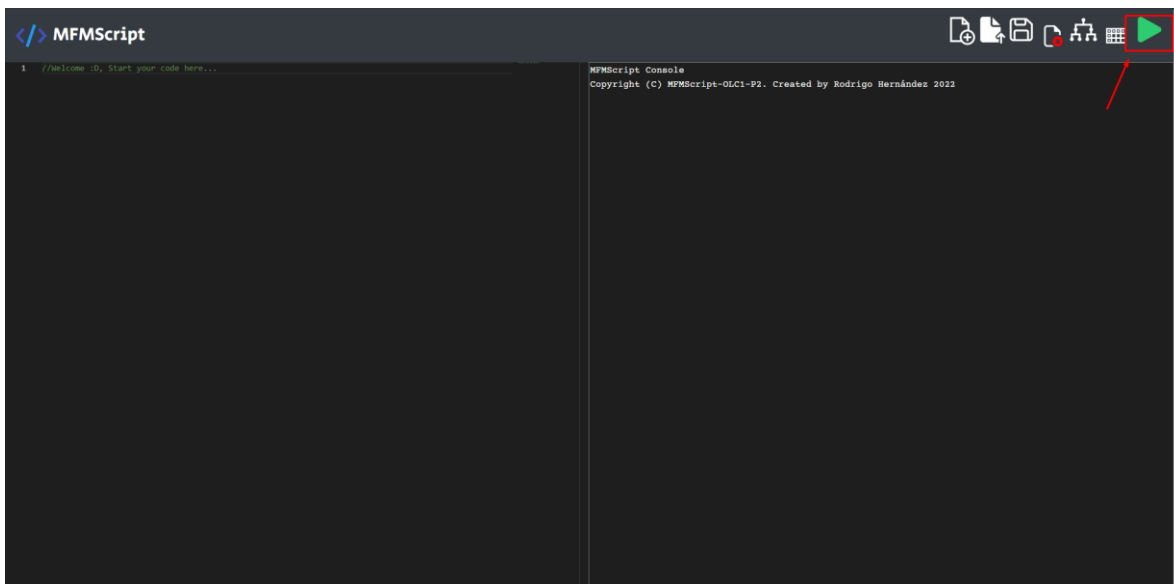
Esta instrucción muestra el contenido de una expresión o valor de una variable. Para poder utilizarla es necesario una expresión o valor de una variable. Al terminar de realizar la impresión se genera un salto de línea por defecto.



```
//SIN SALTO DE LINEA  
print("hola mundo sin salto");  
//CON SALTO DE LINEA  
println("hola con salto");
```

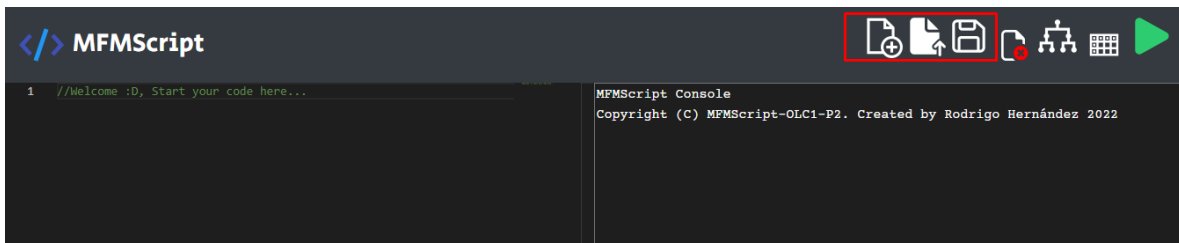
3. Ejecución de código

Al finalizar de generar su pseudocódigo, primero tiene que ejecutarlo en el siguiente botón:



4. Archivo

Este apartado se encuentra en la parte superior de la interfaz:



Y cuenta con las siguientes funcionalidades:

a. Crear Archivo

Esta opción le permite limpiar la interfaz y poder ir creando nuevo código.

b. Abrir Archivo

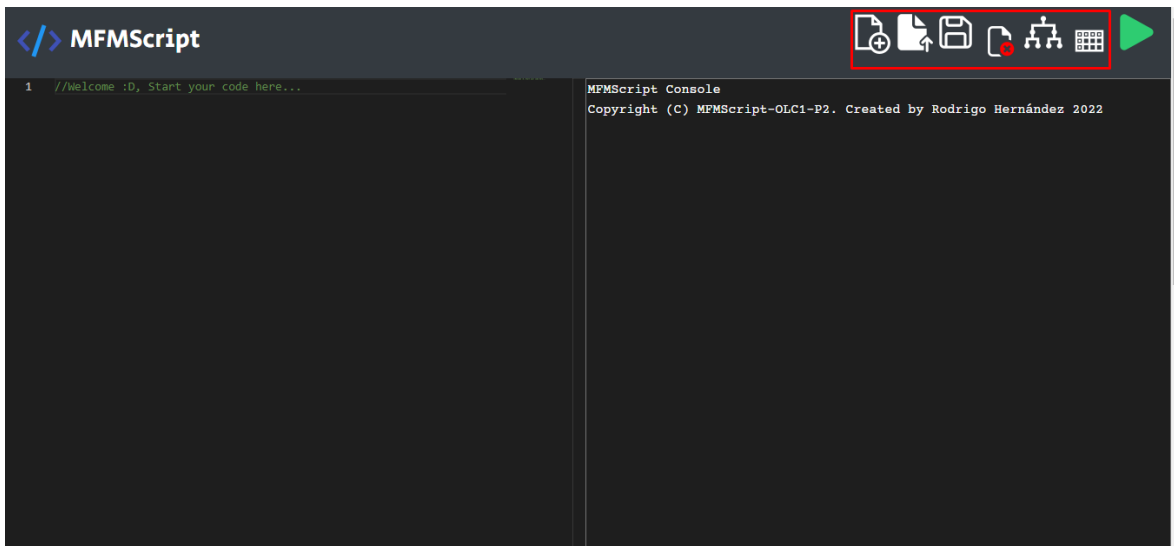
Esta opción le permite únicamente cargar archivos que contengan el lenguaje y la extensión OLC (.olc).

c. Guardar Como

Esta opción le permite guardar el pseudocódigo que tiene escrito en la consola y guardarla en su dispositivo con la extensión olc.

5. Reporte

Este apartado se encuentra en la parte superior de la interfaz.



i. Tabla de Simbolos

Este apartado se dedica a mostrar la tabla de símbolos del interprete.

ii. Errores

Este apartado genera un reporte de una tabla de Errores Léxicos, Semánticos y sintácticos.

iii. AST

Este apartado genera un reporte del Abstract Syntax Tree .