

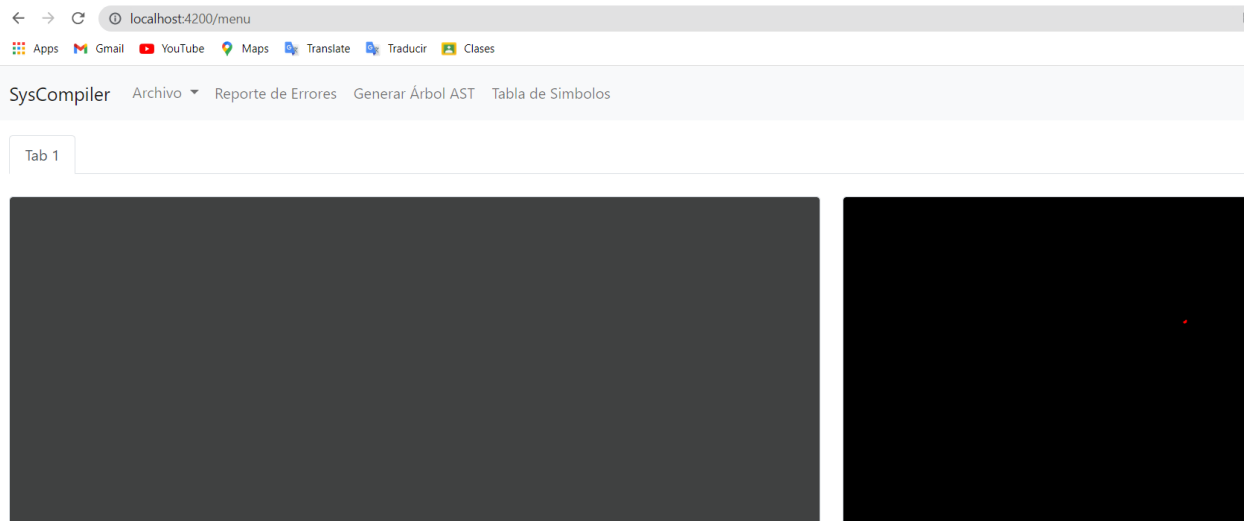


Manual de Usuario SysCompiler

SysCompiler es una pagina web en la que se puede escribir código de un lenguaje determinado y se muestra el resultado de la compilación en una consola al lado del editor de código. SysCompiler utiliza un API para analizar el código de entrada y como respuesta devuelve las salidas que se detectaron en la entrada. Así mismo, el API devuelve la estructura del código que se ejecutó y los posibles errores que se detectaron en uno de los analizadores del compilador. SysCompiler puede luego de ejecutar el código, mostrar una tabla con todos los símbolos detectados en el desarrollo del código, además de una tabla con los errores detectados. Una vez ejecutado el código, SysCompiler tiene la herramienta de generar un AST con la estructura del código analizado.

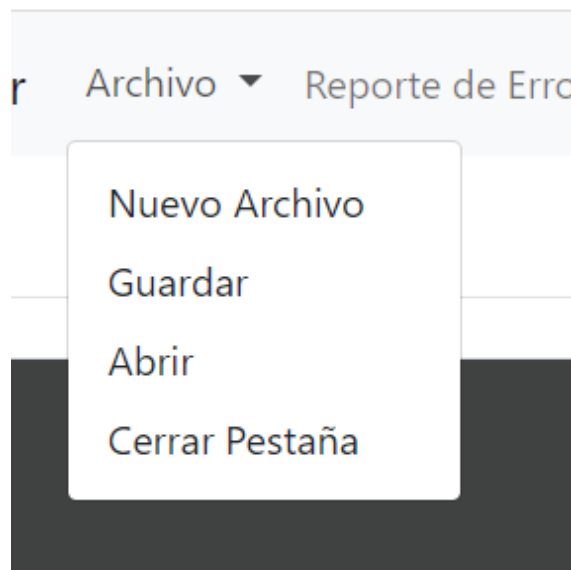
Estructura de la página web

En la pantalla principal de la página de SysCompiler se encuentra una barra con una serie de botones importantes para la ejecución del programa.



▼ Archivo

Este apartado tiene opciones que ayudan a administrar los documentos que servirán como archivos de entrada. Los archivos pueden ser de tipo txt y se desplegará su contenido en el editor de código.



El botón de "Nuevo Archivo" abrirá una nueva pestaña en el editor en donde se puede editar un nuevo código.

El botón de "Guardar" guardará el archivo actualmente abierto en la máquina local.

El botón de "Abrir" abrirá un selector de archivos de la máquina local para mostrar su contenido en el editor de texto actual o en uno nuevo.

El botón de "Cerrar Pestaña" eliminará la pestaña actualmente abierta con todo y el código en ella.

▼ Reporte de Errores

Este botón se ejecuta una vez se haya ejecutado el código presente en la pestaña para desplegar una tabla de errores en la parte inferior del editor.

Tabla de Errores

#	Tipo	Mensaje	Linea	Columna
1	Lexico	@	45	0
2	Sintactico	No coincide el numero de parametros para la funcion funcionesEspecialesYNativas	44	11

▼ Reporte de Símbolos

Este botón se ejecuta una vez se haya ejecutado el código presente en la pestaña para desplegar una tabla de todos los símbolos declarados durante la ejecución del código de entrada en la parte inferior del editor.

Tabla de Simbolos

Identificador	Tipo Símbolo	Tipo	Entorno	Linea	Columna
funcionesEspecialesYNativas	metodo	-	-	1	0
imprimirListaChar	metodo	-	-	46	0
a	variable	INT	Metodo funcionesEspecialesYNativas	2	4
b	variable	DOUBLE	Metodo funcionesEspecialesYNativas	10	4
c	variable	DOUBLE	Metodo funcionesEspecialesYNativas	14	4
cc	variable	DOUBLE	Metodo funcionesEspecialesYNativas	17	4
x	variable	STRING	Metodo funcionesEspecialesYNativas	21	4
y	variable	INT	Metodo funcionesEspecialesYNativas	22	4
z	variable	DOUBLE	Metodo funcionesEspecialesYNativas	23	4
xx	variable	CHAR	Metodo funcionesEspecialesYNativas	24	4
yy	variable	BOOLEAN	Metodo funcionesEspecialesYNativas	25	4
cadena	variable	STRING	Metodo funcionesEspecialesYNativas	32	4

▼ AST

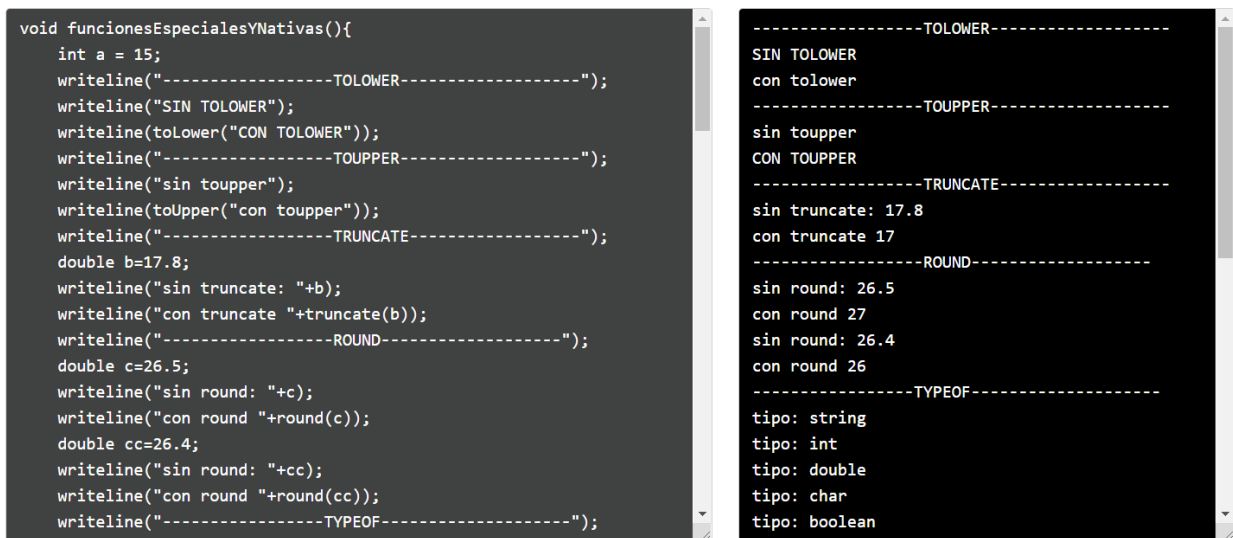
Este botón se ejecuta una vez se haya ejecutado el código presente en la pestaña para desplegar un árbol con la estructura del código en la pestaña actual en la parte inferior del editor.

▼ Ejecutar

El botón ejecutar es el que manda a ejecutar el código en la pestaña actual y permitirá ver el contenido en la consola. Se encuentra en la parte inferior del editor de código.

Consola

La consola sirve para ver las salidas del código ejecutado por el analizador. En ella se pueden ver los resultados de los `writeline` y algunos errores encontrados a lo largo de la ejecución.



```
void funcionesEspecialesYNativas(){
    int a = 15;
    writeline("-----TOLOWER-----");
    writeline("SIN TOLOWER");
    writeline(toLower("CON TOLOWER"));
    writeline("-----TOUPPER-----");
    writeline("sin toupper");
    writeline(toUpper("con toupper"));
    writeline("-----TRUNCATE-----");
    double b=17.8;
    writeline("sin truncate: "+b);
    writeline("con truncate "+truncate(b));
    writeline("-----ROUND-----");
    double c=26.5;
    writeline("sin round: "+c);
    writeline("con round "+round(c));
    double cc=26.4;
    writeline("sin round: "+cc);
    writeline("con round "+round(cc));
    writeline("-----TYPEOF-----");
}
```

```
-----TOLOWER-----
SIN TOLOWER
con tolower
-----TOUPPER-----
sin toupper
CON TOUPPER
-----TRUNCATE-----
sin truncate: 17.8
con truncate 17
-----ROUND-----
sin round: 26.5
con round 27
sin round: 26.4
con round 26
-----TYPEOF-----
tipo: string
tipo: int
tipo: double
tipo: char
tipo: boolean
```

Lenguaje de SysCompiler

El lenguaje de SysCompiler ignora si las palabras reservadas están en mayúsculas o minúsculas, por lo que tanto "FOR" y "for" son palabras reservadas válidas.

▼ Comentarios

Existen dos tipos de comentarios en SysCompiler: de una línea y multilíneas.

Los comentarios de una línea se escriben de la siguiente manera:

```
// Este es un comentario de una linea
```

Los comentarios multilínea se escriben de la siguiente manera

```
/*  
    Este es un comentario multilínea  
*/
```

▼ Tipos de Datos

SysCompiler maneja una variedad de tipos de datos para manejar a sus variables.

- Int: Maneja variables de tipo entero.
- Double: Admite valores numéricos con decimales.
- Boolean: Admite valores que indican verdadero o falso.
- Char: Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples ".
- String: Es un conjunto de caracteres que pueden tener cualquier carácter y se delimita por comillas dobles "".

▼ Operadores Aritméticos

- Suma: Es la operación aritmética que consiste en realizar la suma entre dos o más valores. El símbolo a utilizar es el signo más +
- Resta: Es la operación aritmética que consiste en realizar la resta entre dos o más valores. El símbolo por utilizar es el signo menos -.
- Multiplicación: Operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El signo para representar la operación es el asterisco*.
- División: Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal /.
- Potencia: Es una operación aritmética de la forma a^b donde a es el valor de la base y b es el

valor del exponente que nos indicará cuantas veces queremos multiplicar el mismo número. Por ejemplo 5^3 , $a=5$ y $b=3$ tendríamos que multiplicar 3 veces 5 para obtener el resultado final; $5 \times 5 \times 5$ que da como resultado 125. Para realizar la operación se utilizará el signo $^$.

- Módulo: Es una operación aritmética que obtiene el resto de la división de un numero entre otro. El signo a utilizar es el porcentaje %.
- Negación Unaria: Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original, por ejemplo

▼ Operaciones Relacionales

- Igualación: Compara ambos valores y verifica si son iguales. Se denota por el símbolo " $==$ ".
- Diferenciación: Compara ambos lados y verifica si son distintos. Se denota con el símbolo " \neq ".
- Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo. Se denota con el símbolo " $<$ ".
- Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el lado izquierdo. Se denota con el símbolo " \leq ".
- Mayor que: Compara ambos lados y verifica si el lado izquierdo es mayor que el lado derecho. Se denota con el símbolo " $>$ ".
- Mayor o igual que: Compara ambos lados y verifica si el lado izquierdo es mayor o igual que el lado derecho. Se denota con el símbolo " \geq ".

▼ Operadores Lógicos

- Or: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve verdadero en otro caso retorna falso Se denota con el símbolo " $||$ ".
- And: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve verdadero, en otro caso retorna falso. Se denota con el símbolo " $\&\&$ ".
- Not: Devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá falso, de lo contrario retorna verdadero. Se denota con el símbolo " $!$ ".

▼ Signos de Agrupación

Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por (y).

▼ Caracteres de finalización y encapsulamiento de sentencias

El lenguaje se verá restringido por dos reglas que ayudan a finalizar una instrucción y

encapsular sentencias:

- Finalización de instrucciones: para finalizar una instrucciones se utilizara el signo ;.
- Encapsular sentencias: para encapsular sentencias dadas por los ciclos, métodos, funciones, etc, se utilizará los signos { 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. Las variables podrán ser declaradas global y localmente.

```
<TIPO> identificador;  
<TIPO> id1, id2, id3, id4;  
<TIPO> identificador = <EXPRESION>;  
<TIPO> id1, id2, id3, id4 = <EXPRESION>;  
//Ejemplos  
int numero;  
int var1, var2, var3;  
string cadena = "hola";  
char var_1 = 'a';  
boolean verdadero;  
boolean flag1, flag2, flag3 = true;  
char ch1, ch2, ch3 = 'R'
```

▼ 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 ente paréntesis seguido de una expresión

```
'(<TIPO>)' <EXPRESION>
//Ejemplos
int edad = (int) 18.6; //toma el valor entero de 18
char letra = (char) 70; //tomar el valor 'F' ya que el 70 en ascii es F
double numero = (double) 16; //toma el valor 16.0
```

▼ 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>'--';
//Ejemplos
int edad = 18;
edad++; //tiene el valor de 19
edad--; //tiene el valor 18

int anio=2020;
anio = 1 + anio++; //obtiene el valor de 2022
anio = anio--; //obtiene el valor de 2021
```

▼ 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 dimensión.

```
/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"
```


Para acceder al valor de una posición de un vector, se colocará el nombre del vector seguido de [EXPRESION].

```
<ID> '[' EXPRESION ']'
//Ejemplo de acceso
string vector2[ ] = {"hola", "Mundo"}; //creamos un vector de 2 posiciones de tipo string
string valorPosicion = vector2[0]; //posición 0, valorPosicion = "hola"
```

▼ Listas Dinámicas

Las listas son una estructura de datos que pueden crecer de forma iterativa y pueden almacenar hasta N elementos de un solo tipo; int, double, boolean, char o string.

```
/DECLARACION
'DynamicList' '<' <TIPO> '>' <ID> = new 'DynamicList' '<' <TIPO> '>' '; '
//Ejemplo de declaración
DynamicList<int> lista1 = new DynamicList<int>; //se crea una lista vacía de tipo entero
DynamicList<char> lista1 = new DynamicList<char>; //se crea una lista vacía de tipo char
```

▼ 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 ' (' [<EXPRESION> ] ) ' '{ '
[<INSTRUCCIONES>]
' } '
| 'if' ' (' [<EXPRESION> ] ) ' '{ '
[<INSTRUCCIONES>]
' } ' 'else' '{ '
[<INSTRUCCIONES>]
' } '
| 'if' ' (' [<EXPRESION> ] ) ' '{ '
[<INSTRUCCIONES>]
' } ' 'else' [<IF>]
//Ejemplo de cómo se implementar un ciclo if
if (x <50)
{
    WriteLine("Menor que 50");
    //Más sentencias
}
```

▼ Switch

Switch case es una estructura utilizada para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos if.

```
// EJEMPLO DE SWITCH
int edad = 18;
switch( edad ) {
    Case 10:
        WriteLine("Tengo 10 años.");
        // mas sentencias
        Break;
    Case 18:
        WriteLine("Tengo 18 años.");
        // mas sentencias
    Case 25:
        WriteLine("Tengo 25 años.");
        // mas sentencias
        Break;
    Default:
        WriteLine("No se que edad tengo. :(");
        // mas sentencias
        Break;
}
/* Salida esperada
Tengo 18 años.
No se que edad tengo. :(
*/
```

▼ While

El ciclo o bucle While, es una sentencia que ejecuta una secuencia de instrucciones mientras la condición de ejecución se mantenga verdadera.

```
'while' '(' [<EXPRESION> ']' ')' '{'
[<INSTRUCCIONES>
    '}'
//Ejemplo de cómo se implementar un ciclo while
while (x<100){
    if (x > 50)
    {
        WriteLine("Mayor que 50");
        //Más sentencias
    }
    else
    {
        WriteLine("Menor que 100");
        //Más sentencias
    }
}
```

```

X++;
//Más sentencias
}

```

▼ For

El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella.

```

for '(' ([<DECLARACION>|<ASIGNACION>])';' [<CONDICION>]';' [<ACTUALIZACION>] ')' '{'
[<INSTRUCCIONES>]
'}'
//Ejemplo 1: declaración dentro del for con incremento
for ( int i=0; i<3;i++ ){
WriteLine("i="+i)
//más sentencias
}
/*RESULTADO
i=0
i=1
i=2
*/
//Ejemplo 2: asignación de variable previamente declarada y decremento por asignación
for ( i=5; i>2;i=i-1 ){
WriteLine("i="+i)
//más sentencias
}
/*RESULTADO
i=5
i=4
i=3
*/

```

▼ Do-While

El ciclo o bucle Do-While, es una sentencia que ejecuta al menos una vez el conjunto de instrucciones que se encuentran dentro de ella y que se sigue ejecutando mientras la condición sea verdadera.

```

'do' '{'
[<INSTRUCCIONES>]
'}' 'while' '(' [<EXPRESION>] ')' ';'
//Ejemplo de cómo se implementar un ciclo do-while

```

```

Int a=5;
Do{
  If (a>=1 && a <3){
    WriteLine(true)
  }
  Else{
    WriteLine(false)
  }
  a--;
} while (a>0);
/*RESULTADO
false
false
false
true
true
*/

```

▼ Funciones

Una función es una subrutina de código que se identifica con un nombre, tipo y un conjunto

de parámetros. Para este lenguaje las funciones serán declaradas definiendo primero su

tipo, luego un identificador 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 {}.

```

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

```

```

| [<TIPO>] [<ID>]
//Ejemplo de declaración de una función de enteros
double conversion(double pies, string tipo){
if (tipo == "metro")
{
return pies/3.281;
}
else
{
return -1;
}
}

```

▼ Métodos

Un método también es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros, aunque a diferencia de las funciones estas subrutinas no deben de retornar un valor. Para este lenguaje los métodos serán declarados haciendo uso de la palabra reservada 'void' al inicio, luego se indicará el identificador del método, 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 {}.

```

'void' <ID> '(' [<PARAMETROS> ] ')' '{'
[<INSTRUCCIONES>]
'}'
PARAMETROS -> [<PARAMETROS> ',' [<TIPO>] [<ID>]
| [<TIPO>] [<ID>]
//Ejemplo de declaración de un método

```

```
void holamundo(){
WriteLine("Hola mundo");
}
```

▼ Llamadas

La llamada a una función específica la relación entre los parámetros reales y los formales

y ejecuta la función. Los parámetros se asocian normalmente por posición, aunque, opcionalmente, también se pueden asociar por nombre. Si la función tiene parámetros

formales por omisión, no es necesario asociarles un parámetro real. 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>] ', ' [<ID>]
| [<ID>]
//Ejemplo de llamada de un método

WriteLine("Ejemplo de llamada a método");
holamundo();
/* Salida esperada
Ejemplo de llamada a método
Hola Mundo
*/
//Ejemplo de llamada de una función
WriteLine("Ejemplo de llamada a función");
Int num = suma(6,5); // a = 11
WriteLine("El valor de a es: " + a);
/* Salida esperada
Ejemplo de llamada a función
Aquí puede venir cualquier sentencia :D
El valor de a es: 11
*/
Int suma(int num1, int num2)
{
WriteLine("Aquí puede venir cualquier sentencia :D");
return num1 + num2;
WriteLine("Aquí pueden venir más sentencias, pero no se ejecutarán
por la sentencia RETURN D:"); //WriteLine en una línea
}
```

▼ Writeline

Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter.

```
'WriteLine' '(' <EXPRESION> ')';  
//Ejemplo  
WriteLine("Hola mundo!!");  
WriteLine("Sale compi \n" + valor);  
WriteLine(suma(2,2));
```

▼ Funciones Nativas

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

```
toLowerCase' '(' <EXPRESION> ')';  
//Ejemplo  
string cad_1 = toLowerCase("hola MunDo"); // cad_1 = "hola mundo"  
string cad_2 = toLowerCase("RESULTADO = " + 100); // cad_2 = "resultado = 100"
```

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

```
toUpperCase' '(' <EXPRESION> ')';  
//Ejemplo  
string cad_1 = toUpperCase("hola MunDo"); // cad_1 = "HOLA MUNDO"  
string cad_2 = toUpperCase("resultado = " + 100); // cad_2 = "RESULTADO = 100"
```

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

```
length' '(' <VALOR> ')';  
// En donde <VALOR> puede ser:  
// - lista  
// -vector  
// -cadena
```

- **Truncate:** Esta función recibe como parámetro un valor numérico. Permite eliminar los decimales de un número, retornando un entero.

```
'truncate' '(' <VALOR> ')' ';'
// Ejemplo
int nuevoValor = truncate(3.53); // nuevoValor = 3
int otroValor = truncate(10); // otroValor = 10
double decimal = 15.4654849;
int entero = truncate(decimal); // entero = 15
```

- **Round:** Esta función recibe como parámetro un valor numérico. Permite redondear los números decimales según las siguientes reglas:

```
'round' '(' <VALOR> ')' ';'
// Ejemplo
Double valor = round(5.8); //valor = 6
Double valor2 = round(5.4); //valor2 = 5
```

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

```
typeof' '(' <VALOR> ')' ';'
//Ejemplo
DynamicList<int> lista2 =new DynamicList <int>;
String tipo = typeof(15); // tipo = "int"
String tipo2 = typeof(15.25); // tipo = "double"
String tipo3 = typeof(lista2); // tipo3 = "lista"
```

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

```
toString' '(' <VALOR> ')' ';'
//Ejemplo
String valor = toString(14); // valor = "14"
String valor2 = toString(true); // valor = "true"
```

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

```
'toCharArray' '(' <VALOR> ')' ';'
//Ejemplo
DynamicList<char> caracteres = toCharArray("Hola");
/*
```



```

caracteres [[0]] = "H"
caracteres [[1]] = "o"
caracteres [[2]] = "l"
caracteres [[3]] = "a"
*/

```

▼ Start With

Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia

START WITH para indicar que método o función es la que iniciará con la lógica del programa.

```

start' 'with' <ID> '(' ')' ';'
'start' 'with' <ID> '(' <LISTAVALORES> ')' ';'
LISTAVALORES->LISTAVALORES ',' EXPRESION
| EXPRESION
//Ejemplo 1
void funcion1(){
WriteLine("hola");
}
start with funcion1();
/*RESULTADO
hola
*/
//Ejemplo 2
void funcion2(string mensaje){
WriteLine(mensaje);
}
start with funcion2("hola soy un mensaje");
/*RESULTADO
Hola soy un mensaje
*/

```