

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA  
ESCUELA DE SISTEMAS  
ORGANIZACIÓN DE LENGUAJES COMPILADORES  
ING. MARIO BAUTISTA



LUIS MARIANO MOREIRA  
GARCÍA

202010770

REQUISITOS MINIMOS:

- 100mb de ram
- Intel Pentium
- Navegador Web
- NODE JS INSTALADO

- 

## INSTRUCCIONES DE USO

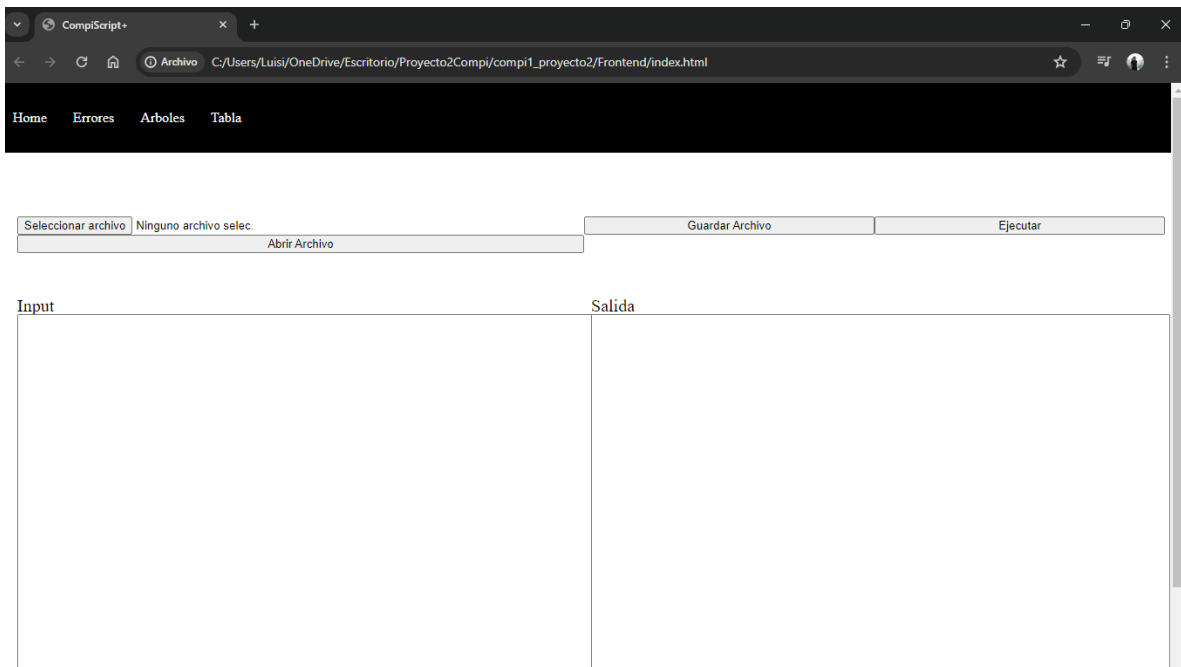
### PARTE 1

#### INICIO




La siguiente aplicación es por medio de cliente servidor:



Es decir que para su funcionamiento es requerido el uso de ambas partes, la parte del cliente esta compuesta por el siguiente archivo html:




Para poder ubicarlo dentro de los archivos en la carpeta llamada frontend:










 compi1_proyecto2	20/04/2024 23:24	Carpeta de archivos
 Frontend	21/04/2024 12:39	Carpeta de archivos
 index	21/04/2024 02:02	Chrome HTML Do... 4 KB

Nota: Para el uso funcional del programa no se debe de eliminar ninguno de los archivos.

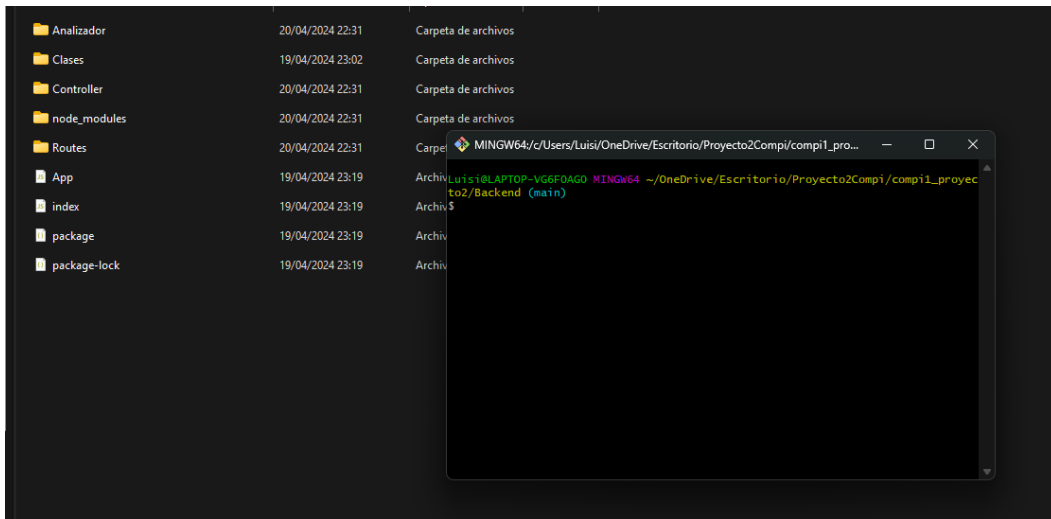
Para hacer uso del servidor, nos dirigimos a la misma carpeta donde esta el frontend:

 Backend	20/04/2024 22:31	Carpeta de archivos
---	------------------	---------------------

Luego en estas carpetas:

 Analizador	20/04/2024 22:31	Carpeta de archivos	
 Clases	19/04/2024 23:02	Carpeta de archivos	
 Controller	20/04/2024 22:31	Carpeta de archivos	
 node_modules	20/04/2024 22:31	Carpeta de archivos	
 Routes	20/04/2024 22:31	Carpeta de archivos	
 App	19/04/2024 23:19	Archivo de origen ...	1 KB
 index	19/04/2024 23:19	Archivo de origen ...	1 KB
 package	19/04/2024 23:19	Archivo de origen ...	1 KB
 package-lock	19/04/2024 23:19	Archivo de origen ...	50 KB

Damos un click derecho y abrimos nuestra terminal (o cmd) de confianza:

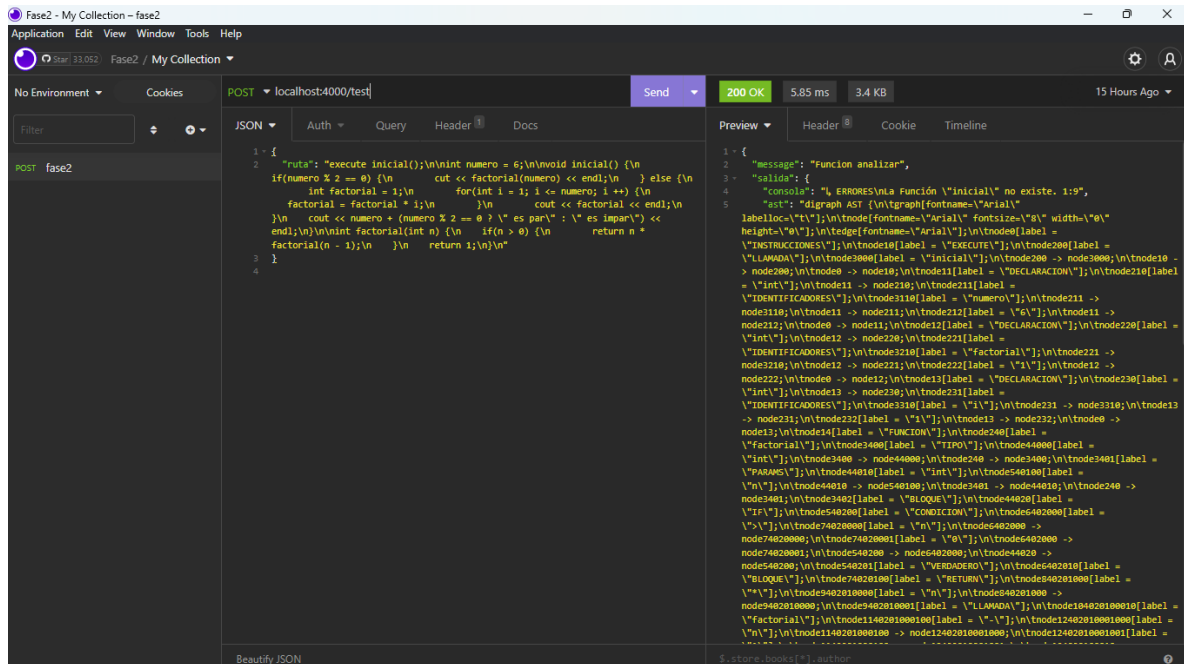


Y para iniciar le damos al comando npm start:

```
MINGW64:/c/Users/Luisi/OneDrive/Escritorio/Proyecto2Compi/compil_pro...  
Luisi@LAPTOP-VG6F0AGO MINGW64 ~/OneDrive/Escritorio/Proyecto2Compi/compil_proyec  
to2/Backend (main)  
$ npm start  
  
> backend@1.0.0 start  
> node index.js  
  
Server en: http://localhost:4000
```

Si la consola se ve de esta forma significa que ya tenemos iniciado nuestro servidor listo para usar.

Nota, si deseas del lado del cliente puedes utilizar una aplicación rest para poder ejecutar tus archivos, nota también que no podrás gozar de las herramientas que la interfaz grafica web te puede proporcionar localhost:4000/test :



PARTE DOS

GRAMATICA

### 5.1 Case Insensitive

El lenguaje es case insensitive por lo que no reconoce entre mayúsculas y minúsculas. Ejemplo:

```
int a = 0;
```

```
iNt A = 0;
```

**Nota:** Ambos casos son lo mismo

### 5.2 Comentarios

Los comentarios son una forma elegante de indicar que función tiene cierta sección del código que se ha escrito, simplemente para dejar un mensaje en específico. El lenguaje deberá soportar dos tipos de comentarios que son los siguientes;

#### 5.2.1 Comentarios de una línea

Este comentario comenzará con `//` y deberá terminar con un salto de línea.

#### 5.2.2 Comentarios multilinea

Este comentario comenzará con `/*` y terminará con `*/`.

```
// Esto es un comentario de una sola línea
```

```
/* Esto es un comentario
```

```
Multilínea */
```

### 5.3 Tipos de Dato

Los tipos de dato que soportará el lenguaje en concepto de un tipo de variable se definen a continuación:

Tipo	Definición	Descripción	Ejemplo	Observaciones	Default
Entero	int	Este tipo de dato aceptará solamente números enteros	1, 50, 100, -120, etc	Del -2147483648 al 2147483647	0
Doble	double	Admite valores numéricos con decimales	1.2, 50.23, 00.34, etc	Se maneja cualquier cantidad de decimales	0.0
Booleano	bool	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
Carácter	char	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples	'a', 'b', 'c', 'E', '1', '&', '\', 'n', etc	En caso de querer escribir comilla simple, se escribirá \ y después la comilla simple \. Si se quiere escribir \ se escribirá \\. Existirá también \n, \t, \r, \".	'\u0000' (carácter 0)
Cadena	std::string	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. " "	"cadena", "- cad"	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: Sirven especialmente en texto para hacer:

Secuencia	Descripción	Ejemplo
\n	Salto de línea	"Hola\nMundo"
\\	Barra invertida	"C:\\miCarpeta"
\"	Comilla doble	"\"esto es una cadena\""
\\t	Tabulación	"\\tEsto es una tabulación"
\'	Comilla simple	"\'Estas son comillas simples\'"



## 5.5 Operadores Aritméticos

### 5.5.1 Suma

Es la operación aritmética que consiste en realizar la suma entre dos o más valores. Para esta se utiliza el signo más (+).

#### Especificaciones de la operación suma

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

+	Entero	Doble	Boolean	Carácter	Cadena
Entero	Entero	Doble	Entero	Entero	Cadena
Doble	Doble	Doble	Doble	Doble	Cadena
Boolean	Entero	Doble			Cadena
Carácter	Entero	Doble		Cadena	Cadena
Cadena	Cadena	Cadena	Cadena	Cadena	Cadena

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

### 5.5.2 Resta

Es la operación aritmética que consiste en realizar la resta entre dos o más valores. Para esta se utiliza el signo menos (-).

#### Especificaciones de la operación resta

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-	Entero	Doble	Boolean	Carácter	Cadena
Entero	Entero	Doble	Entero	Entero	
Doble	Doble	Doble	Doble	Doble	
Boolean	Entero	Doble			
Carácter	Entero	Doble			
Cadena					

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

### 5.5.3 Multiplicación

Es la 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 (\*).

#### Especificaciones de la operación multiplicación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

*	Entero	Doble	Boolean	Carácter	Cadena
Entero	Entero	Doble		Entero	
Doble	Doble	Doble		Doble	
Boolean					
Carácter	Entero	Doble			
Cadena					

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

#### 5.5.4 División

Es la 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 (/).

##### Especificaciones de la operación división

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

/	Entero	Doble	Boolean	Carácter	Cadena
Entero	Doble	Doble		Doble	
Doble	Doble	Doble		Doble	
Boolean					
Carácter	Doble	Doble			
Cadena					

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

#### 5.5.5 Potencia

Es una operación aritmética de la forma **pow(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 pow(5, 3), a=5 y b=3 tendríamos que multiplicar 3 veces 5 para obtener el resultado final; 5x5x5 que da como resultado 125.

##### Especificaciones de la operación potencia

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

pow	Entero	Doble	Boolean	Carácter	Cadena
Entero	Entero	Doble			
Doble	Doble	Doble			
Boolean					
Carácter					
Cadena					

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

#### 5.5.6 Módulo

Es una operación aritmética que obtiene el resto de la división de un número entre otro. Para realizar la operación se utilizará el signo (%).

##### Especificaciones de la operación módulo

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

%	Entero	Doble	Boolean	Carácter	Cadena
Entero	Doble	Doble			
Doble	Doble	Doble			
Boolean					
Carácter					
Cadena					

**Nota:** Cualquier otra combinación no especificada en esta tabla es inválida y será considerado un error de tipo semántico.

### 5.5.7 Negación Unaria

Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original. Se utiliza el símbolo menos (-).

#### Especificaciones de la operación negación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-exp	Resultado
Entero	Entero
Doble	Doble
Boolean	
Carácter	
Cadena	

**Nota:** Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

### Operadores Relacionales:

#### Observaciones:

- Se pueden realizar operaciones relacionales entre: entero-entero, entero-doble, entero-carácter, doble-entero, doble-carácter, carácter-entero, carácter-doble, carácter-carácter y cualquier otra operación relacional entre entero, doble y carácter.
- Operaciones como cadena-carácter, es error semántico, a menos que se utilice `toString` en el carácter.
- Operaciones relacionales entre booleanos es válida.

Operador	Descripción	Ejemplo
<code>==</code>	Igualación: compara ambos valores y verifica si son iguales - Iguales $\rightarrow$ True - No iguales $\rightarrow$ False	<code>1==1</code> <code>"hola" == "hola"</code> <code>25.654 == 54.34</code>
<code>!=</code>	Diferenciación: compara ambos lados y verifica si son distintos - Iguales $\rightarrow$ False - No iguales $\rightarrow$ True	<code>1 != 2</code> <code>var1 != var2</code> <code>50 != 30</code>
<code>&lt;</code>	Menor que: compara ambos lados y verifica si el derecho es mayor que el izquierdo. - Derecho mayor $\rightarrow$ True - Izquierdo mayor $\rightarrow$ False	<code>25.5 &lt; 30</code> <code>54 &lt; 25</code> <code>50 &lt; 'F'</code>
<code>&lt;=</code>	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo. - Derecho mayor o igual $\rightarrow$ True - Izquierdo mayor $\rightarrow$ False	<code>25.5 &lt;= 30</code> <code>54 &lt;= 25</code> <code>50 &lt;= 'F'</code>
<code>&gt;</code>	Mayor que: compara ambos lados y verifica si el izquierdo es mayor que el derecho. - Derecho mayor $\rightarrow$ False - Izquierdo mayor $\rightarrow$ True	<code>25.5 &gt; 30</code> <code>54 &gt; 25</code> <code>50 &gt; 'F'</code>
<code>&gt;=</code>	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho. - Derecho mayor $\rightarrow$ False - Izquierdo mayor o igual $\rightarrow$ True	<code>25.5 &gt;= 30</code> <code>54 &gt;= 25</code> <code>50 &gt;= 'F'</code>

### Operador ternario:

**<CONDICIÓN> '?' <EXPRESION> ':' <EXPRESION>**

```
int edad = 18;  
bool banderaEdad = edad > 17 ? true : false;
```

## Operadores Lógicos:

Operador	Descripción	Ejemplo	Observaciones
	OR: compara expresiones lógicas y si al menos una es verdadera, entonces devuelve verdadero y en otro caso retorna falso.	bandera    5<2 Devuelve true	bandera es true
&&	AND: compara expresiones lógicas y si ambas son verdaderas, entonces devuelve verdadero y en otro caso retorna falso.	flag && "hola" =="hola" Devuelve true	flag es true
!	NOT: devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá false, de lo contrario retorna verdadero	!var Devuelve false	var es true

- **Finalización de instrucciones:** para finalizar una instrucción se utilizará el signo ;
- **Encapsular sentencias:** para encapsular sentencias dadas por ciclos, métodos, funciones, etc, se utilizará los signos { y }.

```
// Ejemplo de finalización de instrucciones
int edad = 18;
// Ejemplo de encapsulamiento de sentencias
if(1==1){
    int a = 10;
    int b = 20;
}
```

### 5.13 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.

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

## Incremento y decremento:

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

Vectores:

// Declaración de tipo 1

<TIPO> <ID> '[' ']' = new <TIPO> '[' <EXPRESION> ']' ';' ;

<TIPO> <ID> '[' '[' ']' '[' ']' = new <TIPO> '[' <EXPRESION> ']' '[' <EXPRESION> ']' ';' ;

// Declaración de tipo 2

<TIPO> <ID> '[' ']' = '[' <LISTAVALORES> ']' ';' ;

#### 5.15.1.2 Acceso a vectores

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

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

// Ejemplos

std::string vector3[] = {"Hola", "Mundo"};

std::string valor3 = vector3[0]; // Almacena el valor "hola"

int vector4 [][] = [ [1, 2], [3, 4] ];

int valor4 = vector4[0][0]; // Almacena el valor 1

#### 5.15.1.3 Modificación de Vectores

Para modificar el valor de una posición de un vector, se debe colocar el nombre del vector seguido de '[' EXPRESION ']' = EXPRESION;

Observaciones:

- A una posición de un vector se le puede asignar el valor de otra posición de otro vector o del mismo vector.
- A una posición de un vector se le puede asignar el valor de una posición de una lista.

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

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

#### 5.16.1 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.

##### 5.16.1.1 if

```
if '(' <EXPRESION> ')' '{  
    [ <INSTRUCCIONES> ]  
}'  
|  
if '(' <EXPRESION> ')' '{  
    [ <INSTRUCCIONES> ]  
}' 'else' '{  
    [ <INSTRUCCIONES> ]  
}'  
|  
if '(' <EXPRESION> ')' '{  
    [ <INSTRUCCIONES> ]  
}' 'else' <IF>  
  
// Ejemplo  
if(x<50){  
    //sentencias  
}
```

##### 5.16.1.2 if else

```
// Ejemplo  
if(x<50){  
    //sentencias  
} else{  
    //sentencias  
}
```

##### 5.16.1.3 else

```
// Ejemplo  
if(x>50){  
    //sentencias  
} else if(x<50 && x>0){  
    //sentencias  
} else {  
    //sentencias  
}
```

### 5.16.2.1 Switch

Estructura principal del switch, donde se indica la expresión a evaluar.

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

### 5.16.2.2 Case

Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch

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

### 5.17.1 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  
while(x<100){  
    //sentencias  
}
```

### 5.17.2 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.

Observaciones:

- Para la actualización de la variable del ciclo for se puede utilizar
  - **Incremento | Decremento:** `i++` | `i--`
  - **Asignación:** como `i = i+1`, `i = i-1`, etc, es decir, cualquier tipo de asignación
- Dentro pueden venir N instrucciones

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

### 5.17.3 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.

Observaciones:

- Dentro pueden venir N instrucciones

```
'do' '{'
[ <INSTRUCCIONES> ]
}' 'while' '(' <EXPRESION> ')' ';'
//Ejemplo
do{
    //sentencias
}while(x<100)
```



### 5.18.2 Continue

La sentencia 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' ';'

//Ejemplo
for(int i = 0; i<5; i++){
    if (i==2){
        continue; // me salte 'mas sentencias' en i = 2
    }
    // mas sentencias
}
```

### 5.18.3 Return

La sentencia 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> ';'

//Ejemplos
for(int i = 0; i<5; i++){
    if (i==2){
        return;
    }
}

for(int i = 0; i<5; i++){
    if (i==2){
        return i;
    }
}
```

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

```
PARAMETROS → PARAMETROS ',' <TIPO> <ID>
            | <TIPO> <ID>
```

//Ejemplo

```
int conversion (double size, std::string tipo){
    if(tipo=="metro"){
        return size/3*3.281;
    } else{
        return -1;
    }
}
```

```
'void' <ID> '(' [<PARAMETROS>] ')' '{'
  [<INSTRUCCIONES>]
'}
```

```
PARAMETROS → PARAMETROS ',' <TIPO> <ID>
            | <TIPO> <ID>
```

//Ejemplo

```
void hola_mundo (){
    cout << "hola mundo" ;
}
```

```

LLAMADA → [<ID>] '(' [<PARAMETROS_LLAMADA>] ')'
          | [<ID>] '(' ' '

PARAMETROS_LLAMADA → PARAMETROS_LLAMADA ',' <TIPO> <ID>
                    | <TIPO> <ID>

//Ejemplo
void hola_mundo () {
    cout << "hola mundo" ;
}
hola_mundo();

int conversion (double size, std::string tipo) {
    if (tipo == "metro") {
        return size / 3 * 3.281;
    } else {
        return -1;
    }
}

int resultado = conversion(58.5, "metro");

```

### 5.21 Función cout

Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter. Al utilizar la palabra reservada 'endl' se permitirá realizar un salto de línea al final del contenido

```

'cout' << <EXPRESION> ';'
'cout' << <EXPRESION> << 'endl' ';'

```

### 5.22 Función tolower

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

```

'tolower' '(' <EXPRESION> ')' ';'

```

### 5.23 Función toupper

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

```

'toupper' '(' <EXPRESION> ')' ';'

```

---

### 5.24 Función 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 a 0.5, se aproxima al entero superior.
- Si el decimal es menor que 0.5, se aproxima al número inferior.

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

### 5.25 Funciones nativas

#### 5.25.1 Length

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

Observación:

- Si se usa para otro parámetro de tipo de dato no especificado, se considera un error semántico.

```
<EXPRESSION> '.' 'length' '(' ')' ;'  
//EXPRESSION puede ser cadenas o vectores
```

#### 5.25.2 Typeof

Esta función retorna una cadena con el nombre del tipo de dato evaluado.

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

#### 5.25.3 ToString

Esta función permite convertir un valor de tipo numérico o bool a texto.

Observación:

- Si se usa para otro parámetro de tipo de dato no especificado, se considera un error semántico.

```
'std' ':' ':' 'toString' '(' <EXPRESSION> ')' ;'
```

### 5.26 Función Execute

Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia EXECUTE para poder indicar qué método o función es la que iniciará con la lógica del programa.

```
'execute' <ID> '(' ')' ';' ;
```

```
'execute' <ID> '(' [<PARAMETROS>] ')' ';' ;
```

## PARTE TRES

### USO DE LA APLICACIÓN WEB

The screenshot shows a web application interface. At the top is a black navigation bar with white text links: "Home", "Errores", "Arboles", and "Tabla". Below the navigation bar is a main content area. At the top of this area is a control bar with three buttons: "Seleccionar archivo", "Guardar Archivo", and "Ejecutar". Below these buttons is a text input field containing the text "Ninguno archivo selec.". Below the input field is a button labeled "Abrir Archivo". The main content area is divided into two large, empty rectangular boxes. The left box is labeled "Input" and the right box is labeled "Salida".

Una vez hayamos seleccionado un archivo aparecerá así:

The screenshot shows the same web application interface as before, but with a file selected. The "Seleccionar archivo" button is now highlighted, and the text "Archivo3.sc" is displayed next to it. The "Abrir Archivo" button is still visible below the input field.

Y al darle Abrir Archivo:

## Input

```
/*
    DECLARAMOS UN VECTOR DE 15 POSICIONES
    SE IMPRIMIR? Y POSTERIORMENTE SE ORDENAR?
*/
int vectorNumeros[] = new int[15];

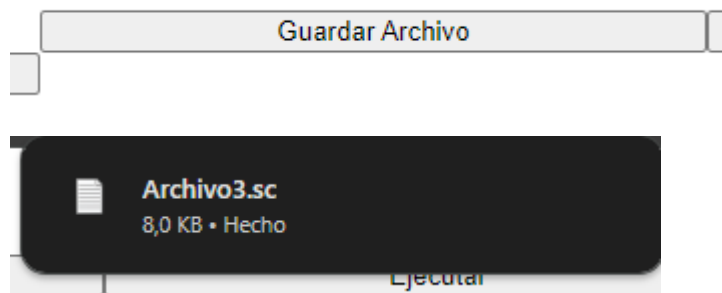
/*
    DECLARAMOS UNA LISTA
*/
std::string frase[] = new std::string[17];

void Hanoi(int discos, int origen, int auxiliar, int destino) {
    if (discos == 1) {
        cout << "Mover disco de " + origen + " a " + destino << endl;
    } else {
        Hanoi(discos - 1, origen, destino, auxiliar);
        cout << "Mover disco de " + origen + " a " + destino << endl;
        Hanoi(discos - 1, auxiliar, origen, destino);
    }
}

void imprimirVector(){
    for (int i = 0; i < vectorNumeros.length(); i++) {
        cout << "vectorNumeros[" + i + "] = " + vectorNumeros[i] << endl;
    }
}
```

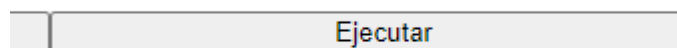
Aparecerá el archivo que hayamos creado.

Si le damos al botón de guardar:



Se descargará el nuevo archivo.

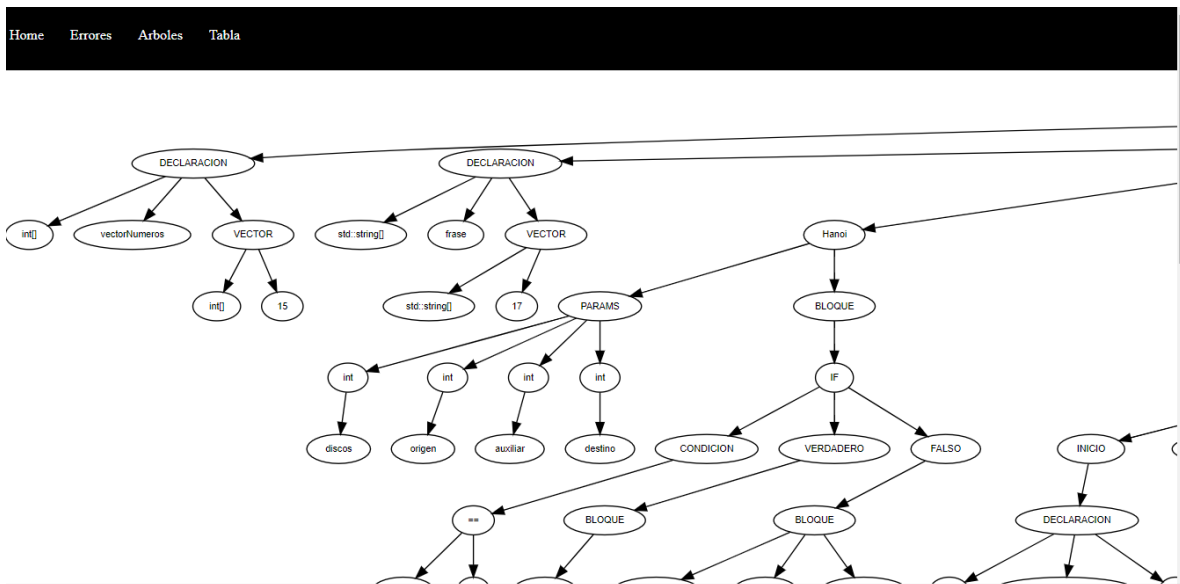
Ahora para ejecutar:



## Salida

```
=====ARCHIVO 3=====1=====
*****SECCION DE VECTORES*****
---Vector Desordenado---
vectorNumeros[0] = 200
vectorNumeros[1] = 26
vectorNumeros[2] = 1
vectorNumeros[3] = 15
vectorNumeros[4] = 167
vectorNumeros[5] = 0
vectorNumeros[6] = 76
vectorNumeros[7] = 94
vectorNumeros[8] = 25
vectorNumeros[9] = 44
vectorNumeros[10] = 5
vectorNumeros[11] = 59
vectorNumeros[12] = 95
vectorNumeros[13] = 10
vectorNumeros[14] = 23
-----Vector Ordenado-----
vectorNumeros[0] = 0
vectorNumeros[1] = 1
vectorNumeros[2] = 5
vectorNumeros[3] = 10
vectorNumeros[4] = 15
vectorNumeros[5] = 23
vectorNumeros[6] = 25
vectorNumeros[7] = 26
```

Se nos carga la salida y si le damos arboles:



Podremos ver el ast generado.

Para ver nuestra tabla de símbolos:



Home	Errores	Arboles	Tabla			
#	Identificador	Tipo de Declaracion	Tipo de Dato	Entorno	Linea	Columna
1	vectornumeros	Variable	int[]	Global	5	1
2	frase	Variable	std::string[]	Global	10	1
3	hanoi	Método	void	Global	12	1
4	imprimirvector	Método	void	Global	22	1
5	bubblesort	Método	void	Global	28	1
6	indicelista	Variable	int	Global	42	1
7	agregarvalorlista	Método	void	Global	43	1
8	imprimirlista	Método	void	Global	48	1
9	mensaje volteado	Función	std::string	Global	54	1
10	par o impar	Método	void	Global	64	1
11	par	Función	int	Global	72	1
12	impar	Función	int	Global	70	1
Home	Errores	Arboles	Tabla			
Error #	Tipo de error	Descripcion	Linea	Columna		
Total de errores: 0						
Luis Mariano Moreira García 202010770						

En caso de tener errores aquí se nos va a notificar.