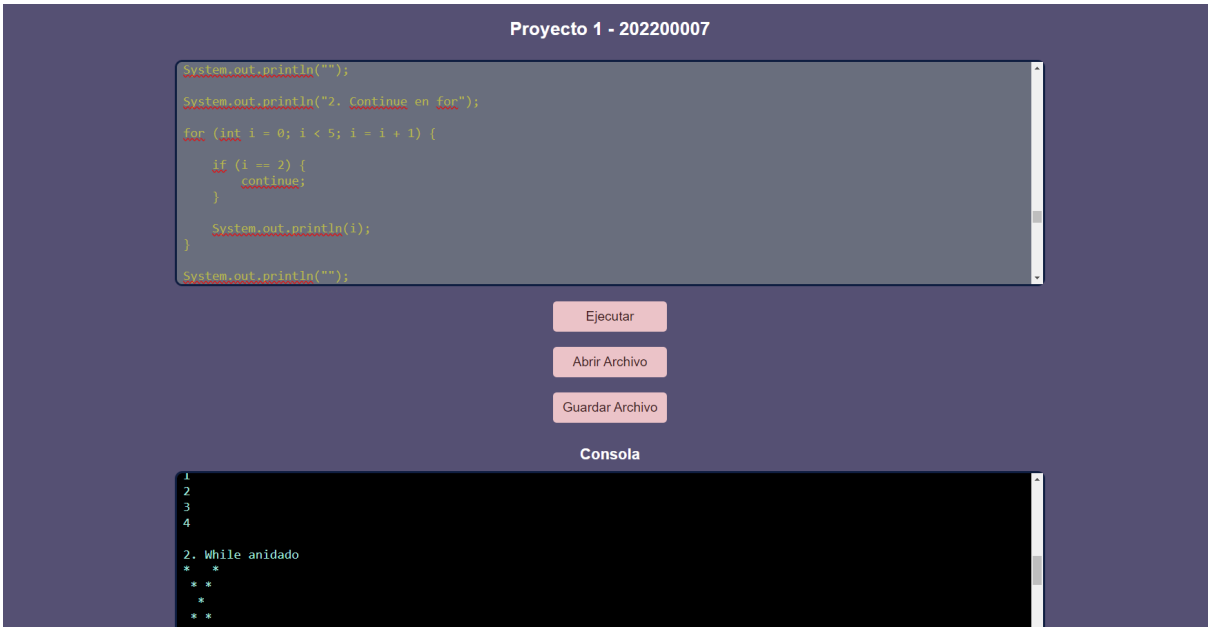
Natalia Mariel Calderón Echeverría
202200007
Organización de Lenguajes y Compiladores 2
Segundo semestre 2024

Manual de Usuario

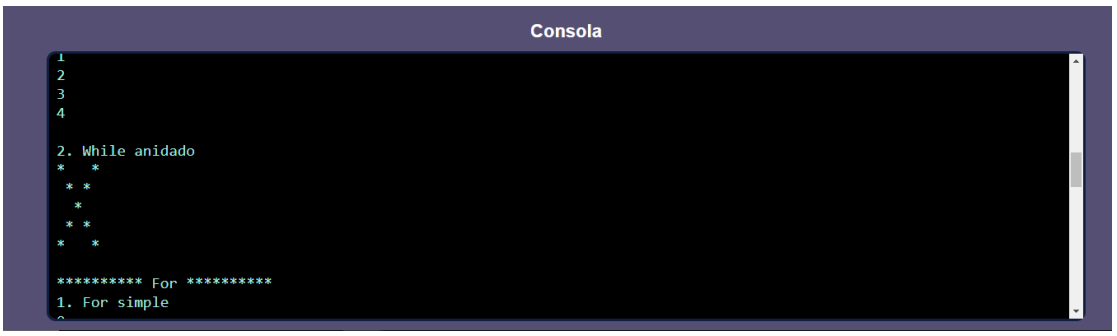
-Menú Inicial

La página principal del intérprete del lenguaje Oakland desarrollado se ve de la siguiente manera:

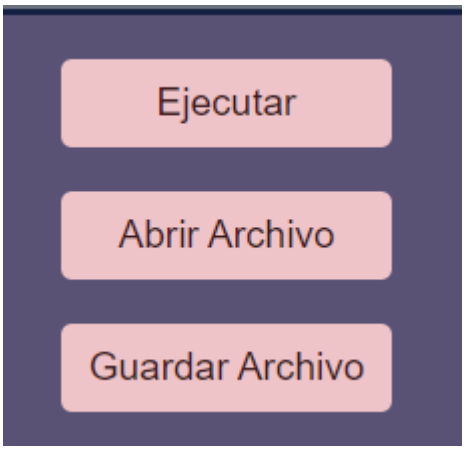


La página principal del proyecto muestra 2 áreas de texto gris y una negra. El área gris corresponde al área en donde se ingresa el código, este área es posible modificar y escribir en ella.

Es en esta área donde se espera que se escriba el código, o el código de un archivo seleccionado se mostrará. Por otro lado, el área negra corresponde a la consola, en donde estará apareciendo el resultado de la ejecución del código proporcionado. Lógicamente, la consola no se puede modificar ni escribir en ella.



Por otro lado, existen tres botones principales: “Ejecutar”, “Abrir Archivo” y “Guardar Archivo”.

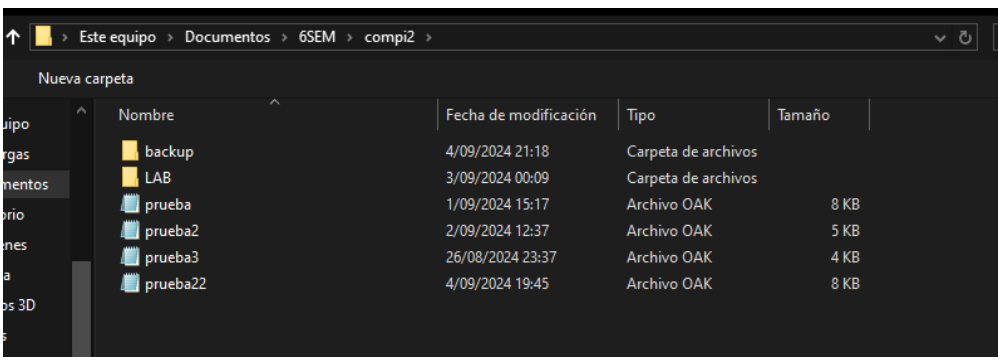


- **Ejecutar**

  El botón ejecutar permite que el codigo que se encuentra escrito en la sección gris (el codigo) sea ejecutado por el intérprete del lenguaje Oakland, dicho resultado será mostrado en la consola posterior a su ejecución. La ejecución puede no ser inmediata, este depende del código y la complejidad de este.

- **Abrir archivo**

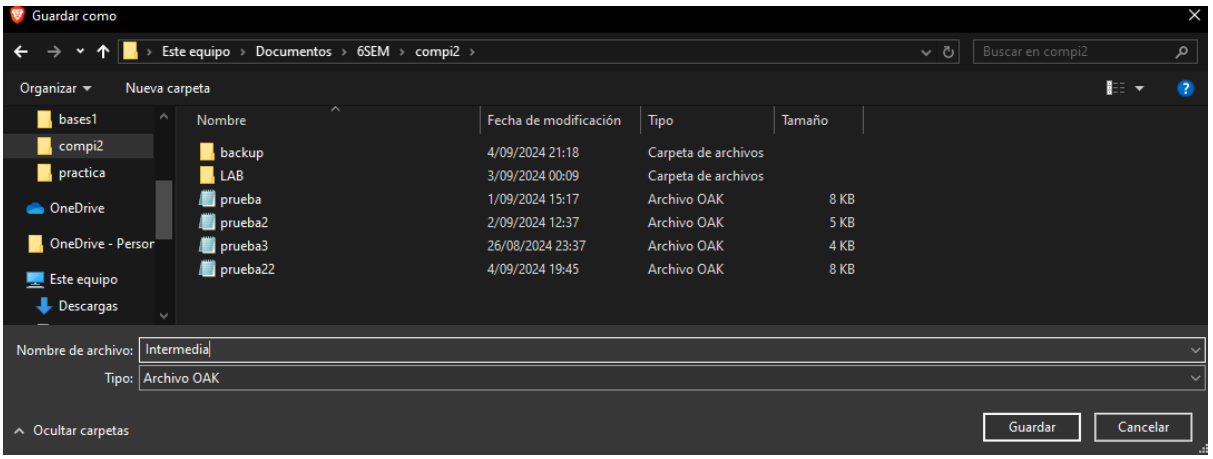
  Al hacer click en el botón abrir archivo se le permite al usuario elegir entre archivos de su computadora para ejecutar, la unica condicion es que dichos archivos sean propias del lenguaje, es decir con extensión .oak



Una vez cargado el archivo este aparecerá en el área de la entrada, este archivo podrá ser modificado antes de su ejecución, si no se desea realizar ninguna modificación, únicamente es necesario proseguir a la ejecución del código.

- **Guardar archivo**

Una vez presionado este botón será posible guardar el código como archivo Oak.



- Reportes

Al momento de ejecutar el código deseado, se generan automáticamente 2 reportes fundamentales relacionados con el código proporcionado. Estos reportes son: Reporte de símbolos (tabla de símbolos) y el reporte de errores.

- ***TABLA DE SÍMBOLOS***

La tabla de símbolos incluye un reporte sobre todos las variables, funciones, arreglos y structs declarados dentro del programa. Esto incluye informacion relevante como: el nombre de la variable, el tipo de la variable, el valor de la variable, el tipo de símbolos (funcion, arreglo, variable o struct) , numero de linea y el número de columna.

ID	Tipo	Valor	Fila	Columna	Estructura
t3_0	string	null	4	24	variable
tr	boolean	true	26	20	variable
blockz	string	ifelse	51	22	variable
prueBaWhile	string	prueBaWhile	112	36	variable
trfal	boolean	false	83	23	variable
a	int	50	87	7	variable
hhi	int	273	141	15	variable

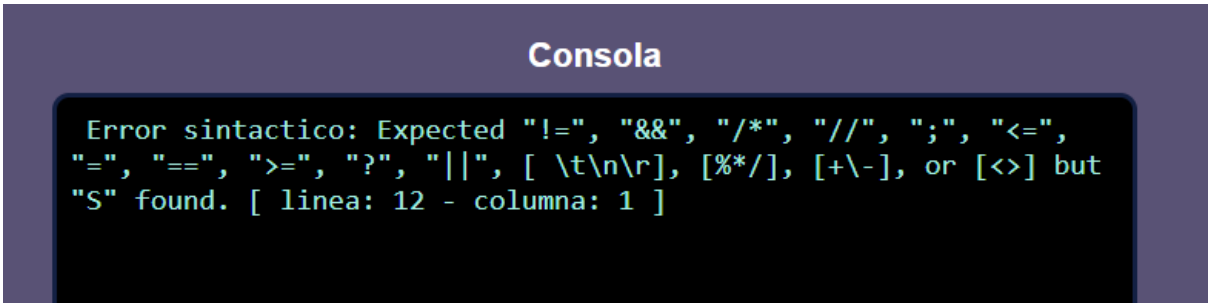
tipobooleaneano	string	boolean	196	45	variable
hy	int	76	200	13	variable
arreglo1	int	1,2,3,4,5,76,7,64,58	201	51	arreglo
MATRIX2	string	h,0,l,a	220	38	arreglo
tec	int	88	229	12	variable
pruebaInt	boolean	false,false,false,false	232	41	Matriz
floatiprueba	float	4.55,8.8,8.98,7.8,15.5,58.15	235	63	arreglo
Segu	float	4.55,8.8,8.98,7.8,15.5,58.15	237	29	arreglo
pruebaFl	float	1.5,2.5,3.5	247	35	arreglo
pruebaIT	int	11,22,23,44,55,66,77,88,88	248	52	Matriz
bul	boolean	false,false,false,true,true,true	249	62	Matriz
signos	char	a,b,c,d,e,f	250	47	arreglo
pruebaDos	float	1.5,2.5,3.5	251	35	arreglo
pruebaTipos	int	15,25,35	252	32	arreglo
cadzz	string	,,,aLAAAAAAAAHHHHH	253	36	arreglo
cpcr	char	a,b,c,d,e,f	263	22	arreglo
NUEVO	int	null	311	32	Matriz
prueba3D	float	1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.1,11.11,92.12,13.13,14.14,15.15,16.16,17.17,18.18	318	134	Matriz
yutu	int	8	329	14	variable
fibonacci	int	null	341	2	funcion
n	int	1	74	7	variableFun
obtenerNumArr	int	null	351	2	funcion
XTipos	int	15,25,35,7,27,17,18,19,11,25	353	47	arreglo

- ***Tabla de errores***

  La tabla de errores generada contiene todos los errores semanticos encontrados a lo largo del código. Los errores encontrados contienen la siguiente información: tipo de error, descripción del error, columna encontrado y linea encontrado.

Tipo	Descripción	Fila	Columna
semantico	Error: Variable t3_0 ya declarada en el entorno actual	16	21
semantico	División por cero no permitida (/)	28	23
semantico	Error de tipado en asignación, el valor declarado no coincide con el tipo de la variable	118	12
semantico	Error: 'if'es una palabra reservada y no puede ser usada como nombre de variable.	158	26
semantico	Error: Variable if no definida	159	22
semantico	Error de tipado en toUpperCase	185	36
semantico	ERROR matriz - error de tipos en la matriz/arreglo o dimensiones no coinciden	266	27
semantico	Error: Variable erTipos no definida	267	27
semantico	ERROR matriz - error de tipos en el arreglo	274	55
semantico	ERROR arreglo- la dimension o el tipo no coinciden	282	24
semantico	ERROR arreglo- la dimension no coinciden	286	23
semantico	ERROR matriz- el tamaño/dimension no coinciden	291	51
semantico	ERROR arreglo- la dimension no coinciden	301	25
semantico	ERROR matrix - la dimension o el tipo no coinciden	307	20
semantico	ERROR arreglo/matrix - error al declarar (tipado o dimensiones no coiciden)	310	30
semantico	Error el arreglo/matrix no puede ser negativo o 0	311	31
semantico	Error: signos dimensiones no coinciden	315	33
semantico	Error de tipos (-=) solo trabaja con float/int	331	12
semantico	Error: Se esperaba un array de una dimension en el foreach	401	2
semantico	Error: Se esperaba un array de tipo float en el foreach	405	2

En el caso de los errores sintácticos estos se exponen directamente en el área de la consola. Se incluye una breve descripción de los tokens encontrados y una lista de los posibles tokens esperados. También se incluye la línea y columna donde estos fueron encontrados



- Lenguaje Oakland

El lenguaje Oakland acepta valores primarios de tipo:

- ❖ Boolean -> true/false
- ❖ Char ->'a'
- ❖ String -> "Hola"
- ❖ Int -> 27
- ❖ float -> 27.7

Estos son los valores primitivos de los cuales esta basado la mayoría de lo que es todo el lenguaje. El lenguaje de programación Oakland es un lenguaje tipado por lo que es muy importante el tipo de dato ya que los tipos de datos siempre deben coincidir. La única conversión implícita que ocurre en este lenguaje es la de int a float.

Por otro lado, también existen los valores compuestos que hacen referencia a:

- Arrays (N dimensionales)
  Al igual que los valores primitivos los Array también cuentan con esos mismos tipos. Una de las características importantes de los arrays en este lenguaje es que todos deben ser del mismo tipo. Es decir, no puede existir un array que tenga valores distintos entre sí, por ejemplo: {1,2,3,4.5,"Error"}

  Los arreglos cuentan con ciertas funciones nativas a ellos estas son:
  - indexOf()

Devuelve el índice de la primera coincidencia que encuentre, si no encuentra ninguna coincidencia retorna -1

- join()

  Crea un string de los elementos del arreglo
- length

  Devuelve un numero int del valor del largo del arreglo

```
System.out.println("FUNCIONES DE ARREGLOS");
////
System.out.println(XTipos);
System.out.println(XTipos.indexOf(27));
System.out.println(XTipos[2]);
System.out.println(XTipos.length);
string HElLo = XTipos.join();
System.out.println(HElLo);

//error
for(float elemento : prueba3D){
```

```
FUNCIONES DE ARREGLOS
15,25,35,7,27,17,18,19,11,25
4
35
10
15253572717181911 25
```

- Structs

  Los structs se definen como una estructura de datos compuestos que permiten una mejor manipulación de la información. Estos funcionan como plantillas que posteriormente se pueden utilizar para declarar instancias de las mismas.

```
struct Persona {
    string nombre;
    int edad;
    float estatura;
};

struct Auto {
    string marca;
    string modelo;
    int anio;
    Persona propietario;
};
```

Los structs cuentan una función nativas esta es:

- Object.Keys

La cual devuelve el nombre de los atributos del struct

```
System.out.println(Object.keys(persona1));
System.out.println(Object.keys(auto1));
```

```
********** Funcion Object.keys **********
nombre, edad, estatura
marca, modelo, anio, propietario
```

- Declaración

  Las variables simples de datos nativos se pueden declarar de 4formas:

  - Con la palabra var seguida del Id deseado para la variable y la expresión que se desea asignar.

    ```
    var t3_0= "nata"+"lia";
    ```

  - El tipo de la variable seguido del Id deseado para la variable y la expresión que se desea asignar (los tipos deben coincidir)

    ```
    int prueBaWhile = 22-(4*3);
    ```

  - El tipo de la variable y el id que se desea asignar. En este caso el valor por defecto de la misma es “null”

    ```
    int a;
    ```

  - Por copia de valores

    ```
    int numero = a;
    ```

    Es necesario aclarar que los nombres de las variables no pueden ser repetidos en un mismo entorno y que el los tipos declarados y asignados deben coincidir, de lo contrario existiría un error de tipo semántico.

Los arreglos al igual que las variables deben coincidir en tipo, el tipo declarado debe ser igual al tipo asignado.

```
float[] pruebaFl = {4.55,8.8,7.7};


int[][] pruebaII = {{1, 2, 3},{4, 5, 6},{7,8,11}};
```

A diferencia de las variables los arreglos también deben coincidir en dimensiones, es decir, si el arreglo declarado es de 2 dimensiones el arreglo asignado también debe serlo. De lo contrario, se tomará como error de tipo semántico.

```
boolean[][] bul = {{true, true, true},{false, false, false}};
char[] signos = {'a', 'b', 'c', 'd','e', 'f'};
float[] pruebaDos = {1.5,2.5,3.5};
int[] pruebaTipos = {15,25,35};
string[] cadzz = {"h","0","l","a"};
```

Los arreglos también se pueden declarar a través de la creación de arreglos vacíos, esta forma de declaración también debe cumplir con el requisito de tipos.

Es decir, esto sería considerado un error, tanto por el tipo como por las dimensiones declaradas:

```
float[][] NUEVO = new int[3];
```

Esta es la forma correcta de declarar un arreglo con un arreglo vacío:

```
float[][] NUEVO = new float[3][3];
```

Una peculiaridad de los arreglos es que pueden ser accedidos y modificados. El acceso y modificación de los mismos sigue las reglas de la mayoría de los lenguajes de programación. Se indica el índice al cual se quiere acceder o el índice del cual se desea modificar.

```
float[][][] prueba3D = {{{1.1,2.2,3.3},{4.4,5.5,6.6}},
{{7.7,8.8,9.9},{10.10,11.11,12.12}},{{13.13,14.14,15.15},
{16.16,17.17,18.18}}};
prueba3D[1][1][2] += 80;
```

Los structs por su parte se declaran y se instancia de manera peculiar. La declaración de un struct solo puede realizarse a traves de la palabra struct seguida del id deseado y el formato y los atributos que se desean agregar al struct, por ejemplo:

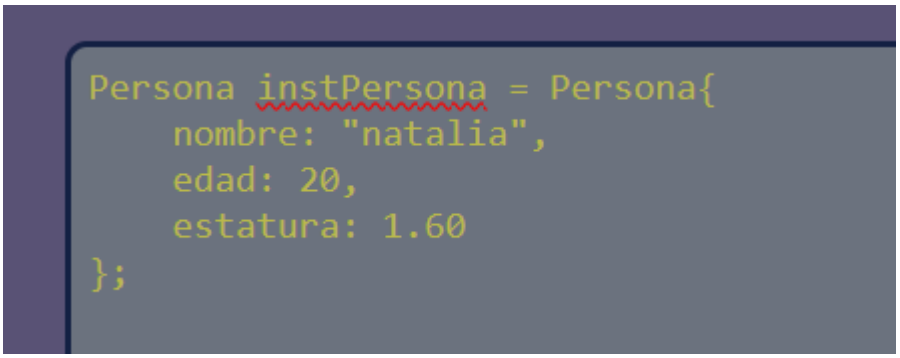
```
struct Persona {
    string nombre;
    int edad;
    float estatura;
};

struct Auto {
    string marca;
    string modelo;
    int anio;
    Persona propietario;
};
```

Sin embargo, un struct puede ser instanciado de 2 maneras, una de ellas es a través de la declaración de este como variable a trabes de la palabra reservada var, seguido del id y el valor que se desea asignar (se debe indicar el struct a utilizar)

```
var instPersona = Persona{
    nombre: "natalia",
    edad: 20,
    estatura: 1.60
};
```

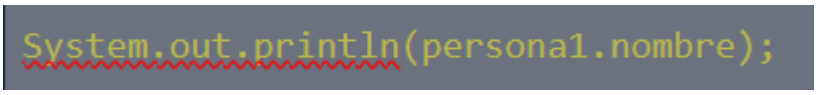
También es posible instanciar un struct a través de la declaracion del tipo de Struct a utilizar, el id que se desea para la instancia y el valor del struct

que se desea asignar. Tanto la declaración como la asignación deben coincidir con el struct con el que se está trabajando.



```
Persona instPersona = Persona{
    nombre: "natalia",
    edad: 20,
    estatura: 1.60
};
```

Al igual que en los arreglos y en las variables es posible acceder a estos y modificarlos a los largo del código, tanto como para acceder y modificarlos es necesaria seguir la siguiente sintaxis: id . atributo



```
System.out.println(persona1.nombre);
```

```
persona1.nombre = "Mariel";
```