Manual De Usuario

Proyecto 1 Organización de Lenguajes y Compiladores 1

Objetivos e Información del Sistema

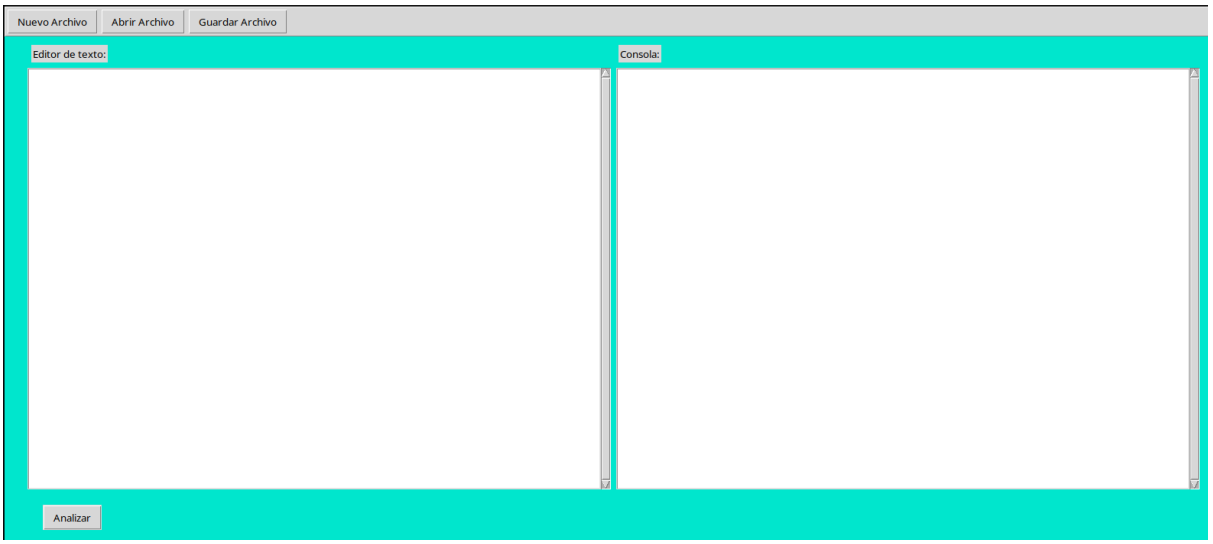
Este Programa con su propia sintaxis, es capaz de interpretar el lenguaje OLCScript que está basado en el lenguaje de alto nivel Typescript. Este lenguaje incorpora varias funciones del lenguaje de alto nivel Typescript , además incorpora otras funcionalidades de alto nivel para hacerlo un lenguaje versátil, moderno y eficiente.

Requisitos del sistema

- Procesador AMD Ryzen 5 3400G o superior
- 16 Gb Ram
- Gráficos Radeon RX 6600 o superior
- Conexión a internet
- Navegador Web
- Java

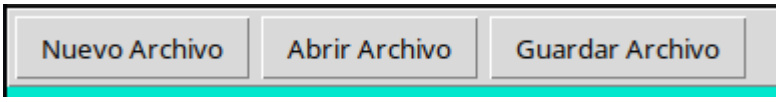
Interfaz

En la interfaz podemos encontrar todos los controles para usar el programa.



Archivos

En este apartado podemos abrir archivos, guardar y guardar como.



Analizador

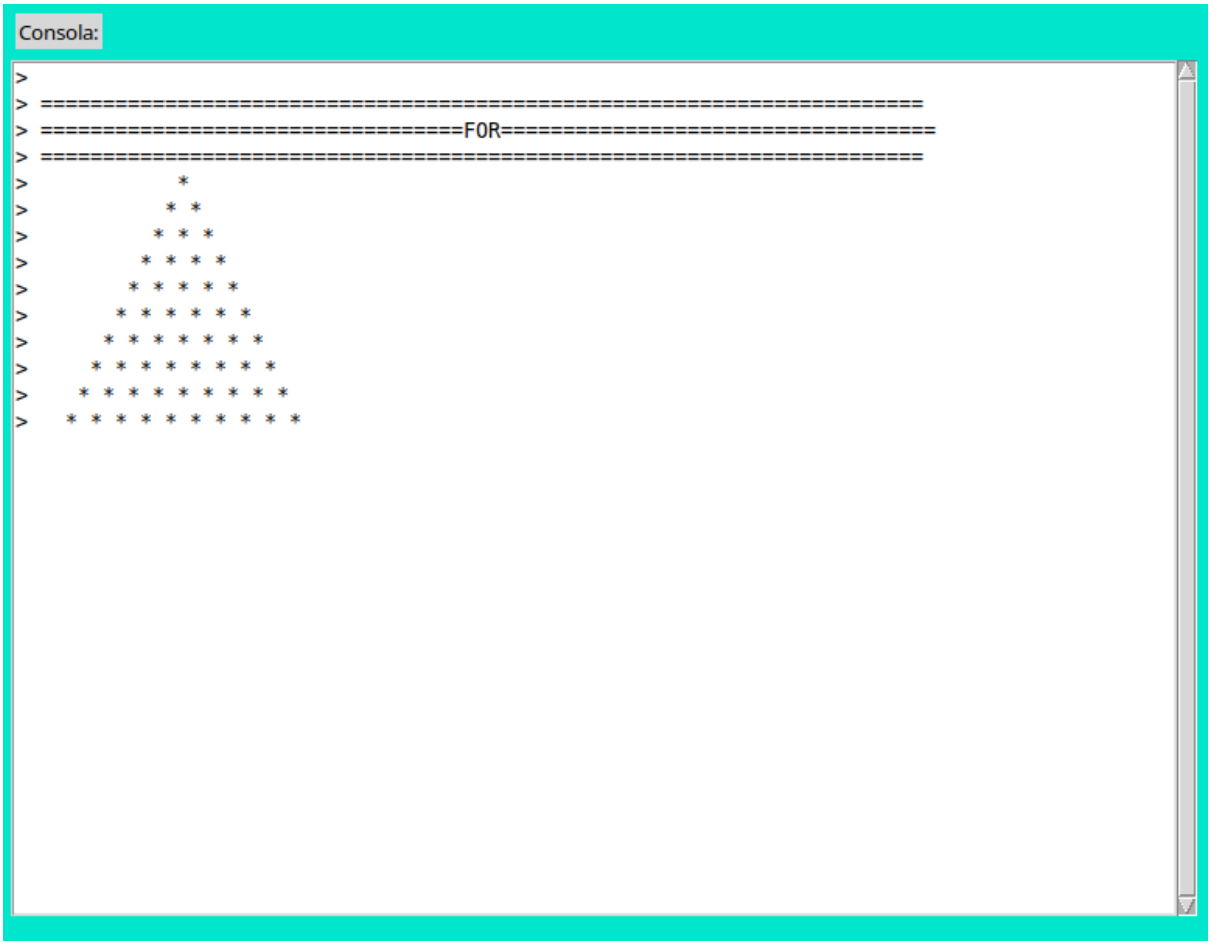
Con este botón podemos analizar el código en el área de texto y ejecutarlo.

Entrada

En este apartado encontraremos un cuadro de texto donde podremos escribir el código que queramos, o editar el de un archivo subido.

Salida

Después de ejecutar el análisis del código escrito, obtendremos la salida en la consola.



Reporte Tokens y Errores

Después de leer el codigo, se generarán los reportes de tokens y de errores en archivos html.

Reporte Tabla Simbolos

Id	Tipo	Tipo Dato	Valor	Ambito	Linea	Columna
i	Variable	Number	10	global_For	0	0
output	Variable	Cadena	* * * * * * * * *	global_For	0	0
j	Variable	Number	2	global_For_For	0	0
k	Variable	Number	10	global_For_For	0	0

Lenguaje Para Expresiones Regulares

El lenguaje a usar es OLCScript, es case sensitive y está inspirado en el lenguaje Typescript, este mismo es conocido por su versatilidad al ser un lenguaje multiparadigma que ha ganado considerable popularidad. OLCScript no sólo hereda las ventajas de

Typescript, sino que también incorpora funcionalidades avanzadas como inferencia de tipos, tipado estático, etc.

Sintaxis Comentarios

```
// Esto es un comentario de una línea
/*
Esto es un comentario multilínea
*/
```

Sintaxis Bloque Sentencias

```
{
// sentencias
}
```

Sintaxis Declaración variables

```
// declaración con tipo y valor
var <identificador> : <Tipo> = <Expresión> ;
// declaración con valor
var <identificador> = <Expresión> ;
// declaración con tipo y sin valor
var <identificador> : <Tipo> ;
```

Sintaxis Operador ternario

```
<CONDICIÓN> ? <EXPRESIÓN> : <EXPRESIÓN> ;
```

Sintaxis If

```
if ( <EXPRESIÓN> ) { <BLOQUE_SENTENCIAS> }
| if ( <EXPRESIÓN> ) { <BLOQUE_SENTENCIAS> } else { <BLOQUE_SENTENCIAS>}
| if ( <EXPRESIÓN> ) { <BLOQUE_SENTENCIAS> } else SI
```

Sintaxis Switch

```
case -> switch (<Expresión>) {
case expr1:
# Declaraciones ejecutadas cuando el resultado de expresión coincide con
el expr1
break
case expr2:
# Declaraciones ejecutadas cuando el resultado de expresión coincide con
el expr2
break
...
case exprN:
# Declaraciones ejecutadas cuando el resultado de expresión coincide con
exprN
break
default:
# Declaraciones ejecutadas cuando ninguno de los valores coincide con el
valor de la expresión
}
```

Sintaxis While

```
WHILE -> while (<Expresión)
<BLOQUE SENTENCIAS>
}
```

Sintaxis For

```
for (var i: number = 1; i <= 5; i++) {
console.log(i);
}
```

Sintaxis break, Continue y return

```
break;
continue;
return;
return 1;
```

Sintaxis Array

```
<Declaracion_array> -> <Tipo_declaracion> Id : <TIPO>[] = <Definicion_array> ->
```

Sintaxis Interface

```
<Def_Interface> -> interface ID { <Lista_Atributos>
}
<Lista_Atributos> -> <Lista_Atributos> ; ID : <TIPO>
| ID : <TIPO>
```

Sintaxis Funciones

```
function func1(): number{
return 1;
}
```