

Laboratorio de Organización de Lenguajes y  
Compiladores 1, Sección P

# PROYECTO OBJ-C

**MANUAL USUARIO**

FECHA: 27/06/2025

Byron Miguel Galicia Hernandez 201907177

Cesar Armando Garcia Franco 202110378

Alvaro Gabriel Ceballos Gil 202300673

Yulianne Nicole López Elias 202300659

## **Objetivos del Sistema**

El sistema permite al usuario cargar, visualizar y editar archivos de código (.objc) en una interfaz web, interpretar y ejecutar inmediatamente el código detectando y reportando errores léxicos, sintácticos y semánticos en tiempo real, mostrar gráficamente el Árbol de Sintaxis Abstracta (AST) y las tablas de símbolos para facilitar la comprensión del flujo de ejecución, gestionar vectores multidimensionales con sus operaciones de declaración, asignación y acceso, así como aplicar funciones nativas (seno, coseno, inverso) y de manipulación de vectores (sort, shuffle), y generar reportes detallados que apoyen el proceso de depuración y aprendizaje.

## Información del Sistema

El sistema es una herramienta educativa y de desarrollo diseñada para facilitar el aprendizaje y la experimentación con conceptos de compiladores e interpretación de lenguajes. A través de una interfaz web intuitiva, permite al usuario cargar archivos de código con extensión `.objc`, visualizar su contenido, editarlo y enviarlo al backend para su análisis. El motor de interpretación, implementado en Python con la biblioteca `PLY`, se encarga de realizar el análisis léxico, sintáctico y semántico, construyendo internamente un Árbol de Sintaxis Abstracta (AST) y generando tablas de símbolos que documentan variables, funciones y vectores declarados.

Una vez procesado el código, el sistema muestra de manera inmediata y clara los resultados de la ejecución: por un lado, la consola virtual despliega los valores producidos por las instrucciones `println`; por otro, los reportes de errores indican el tipo (léxico, sintáctico o semántico), la ubicación (línea y columna) y una breve descripción de cada incidencia. Además, se pueden generar visualizaciones gráficas del AST y de las tablas de símbolos, lo cual es de gran ayuda para entender el flujo del programa y detectar inconsistencias lógicas o estructurales.

El sistema también incorpora soporte para estructuras avanzadas: vectores multidimensionales que pueden declararse, inicializarse con valores literales o mediante funciones (`sort`, `shuffle`) y asignarse por posición, así como funciones matemáticas nativas (`seno`, `coseno`, `inverso`) que admiten tanto literales numéricos como accesos a vectores. Gracias a este conjunto de funcionalidades, el usuario cuenta con un entorno completo para diseñar, depurar y analizar su propio mini-lenguaje, fomentando la comprensión práctica de principios de compilación y ejecución de programas.

## Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

## Inicialización del programa

Para poder acceder a la inicialización del programa se debe estar seguro de tener instalados todas las librerías necesarias para el despliegue, las cuales son:

- **Flask:** micro-framework web para Python, usado como servidor HTTP y para definir rutas.
- **PLY** (Python Lex-Yacc): proporciona las herramientas de análisis léxico y sintáctico.
- **math:** módulo estándar de Python (no requiere instalación adicional) para funciones trigonométricas (seno, coseno, inverso).

Una vez instaladas las librerías y componentes, ingrese a visual studio code y abra el directorio del proyecto.

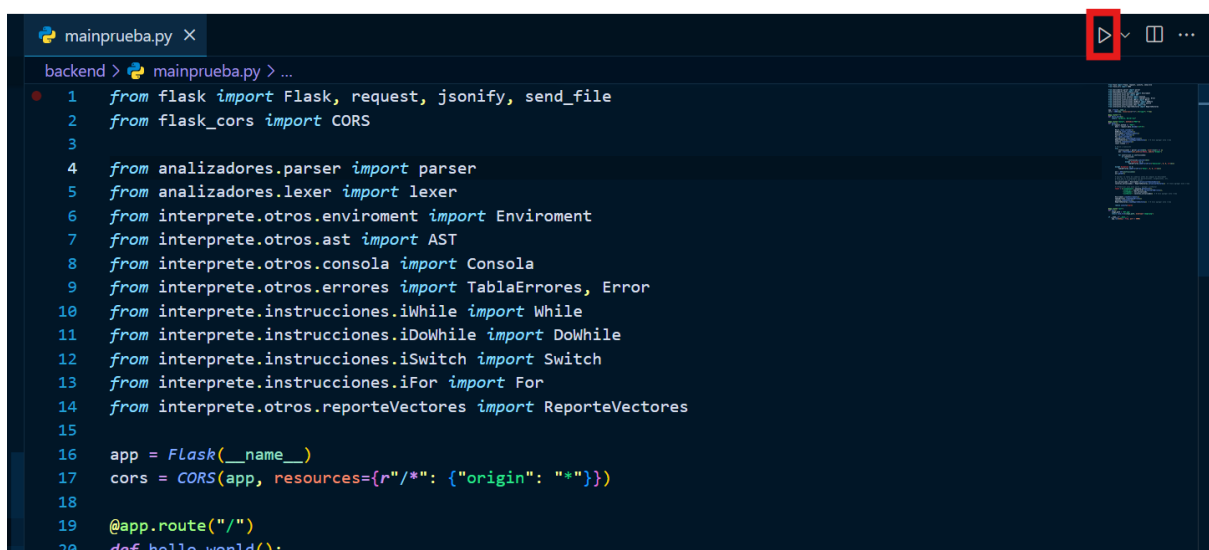
Una vez aquí, abra una nueva terminal y proceda a ejecutar el siguiente comando:



```
PS C:\Users\Nicole\Desktop\OLC1_VJ2025_G5> npm install
```

*Nota Importante: Evite realizar cambios en el código fuente únicamente ejecute el comando:*

*En la clase mainprueba.py ejecute*



```
mainprueba.py X
backend > mainprueba.py > ...
1 from flask import Flask, request, jsonify, send_file
2 from flask_cors import CORS
3
4 from analizadores.parser import parser
5 from analizadores.lexer import lexer
6 from interprete.otros.enviroment import Enviroment
7 from interprete.otros.ast import AST
8 from interprete.otros.consola import Consola
9 from interprete.otros.errores import TablaErrores, Error
10 from interprete.instrucciones.iWhile import While
11 from interprete.instrucciones.iDowhile import Dowhile
12 from interprete.instrucciones.iSwitch import Switch
13 from interprete.instrucciones.iFor import For
14 from interprete.otros.reporteVectores import ReporteVectores
15
16 app = Flask(__name__)
17 cors = CORS(app, resources={r"/*": {"origin": "*"}})
18
19 @app.route("/")
20 def hello_world():
```

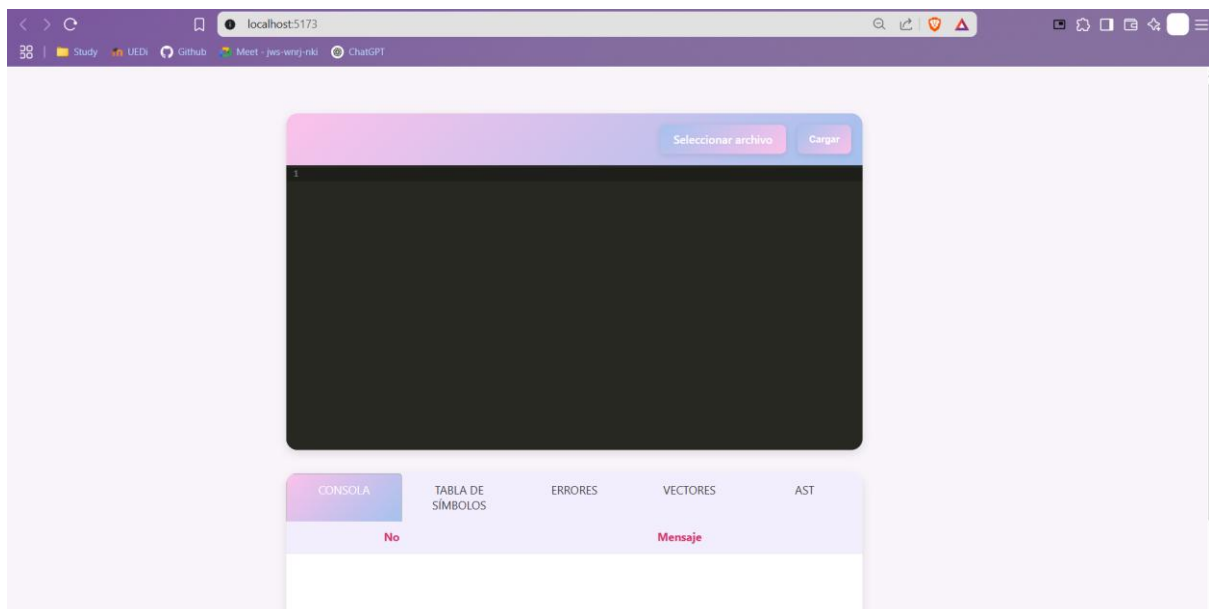
*Abra una terminal en el frontend y escriba npm run dev*

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Nicole\Desktop\OLC1_VJ2025_G5\frontend> npm run dev
```

*Aparecera un link y da ctrl clic*

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
VITE v6.3.5
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

## Ventana del menú principal de la aplicación:



En la parte inferior de la interfaz web encontrarás una vista en pestañas que te permite inspeccionar distintos aspectos de la ejecución de tu programa:

- **Consola**

Aquí se listan las salidas generadas por las instrucciones `println(...)` y otros

mensajes de ejecución (por ejemplo, retornos de funciones). Cada línea aparece numerada y junto a ella el texto resultante.

- **Tabla de Símbolos**

Muestra todas las variables (y, en futuras versiones, procedimientos) que el intérprete ha declarado en el entorno actual. Para cada símbolo verás su nombre, su tipo (INT, FLOAT, BOOL, CHAR, STR, VECTOR...), su valor actual y el ámbito en que fue definido (global o local).

- **Errores**

En esta pestaña se acumulan los errores sintácticos, semánticos y de ejecución detectados durante el análisis o la interpretación. Cada entrada indica el tipo de error, una breve descripción y la línea/columna donde ocurrió.

- **Vectores**

Cuando declaras o modificas vectores, aquí se despliega una tabla especializada que lista, para cada vector, su identificador, tipo de datos, dimensiones, tamaño total, estrategia de almacenamiento y (opcionalmente) su contenido actual.

- **AST**

Finalmente, en “AST” se muestra (o se enlaza) el árbol de sintaxis abstracta generado tras el análisis léxico y sintáctico. Suele representarse gráficamente para ayudar a visualizar la estructura jerárquica de las instrucciones y expresiones de tu programa.

# Ejemplo de ejecución

Seleccionar archivoCargar

```
1 println("-----");
2 println("Vector de una dimension");
3 println("-----");
4
5 Vector<int> ud(3) = [1,2,3];
6
7 println(ud[0]);
8 println(ud[1]);
9 println(ud[2]);
10 println(ud[3]); // Error semantico
11 println(ud[0][1]); // Error semantico
12
```

CONSOLA

TABLA DE SÍMBOLOS

ERRORES

VECTORES

AST

No	Mensaje
0	-----
1	Vector de una dimension

Seleccionar archivoCargar

```
1 println("-----");
2 println("Vector de una dimension");
3 println("-----");
4
5 Vector<int> ud(3) = [1,2,3];
6
7 println(ud[0]);
8 println(ud[1]);
9 println(ud[2]);
10 println(ud[3]); // Error semantico
11 println(ud[0][1]); // Error semantico
12
```

CONSOLA

TABLA DE SÍMBOLOS

ERRORES

VECTORES

AST

Tipo	Descripción	Línea	Columna
Semántico	Índice 3 fuera de rango para la dimensión 1 (0-2)	10	195
Semántico	Error en la expresión de la función println()	10	187
Semántico	El vector tiene 1 dimensiones, pero se proporcionaron 2 índices	11	230
Semántico	Error en la expresión de la función println()	11	222



Seleccionar archivo

Cargar

```

1 println("-----");
2 println("Vector de una dimension");
3 println("-----");
4
5 Vector<int> ud(3) = [1,2,3];
6
7 println(ud[0]);
8 println(ud[1]);
9 println(ud[2]);
10 println(ud[3]); // Error semantico
11 println(ud[0][1]); // Error semantico
12

```

CONSOLA

TABLA DE SÍMBOLOS

ERRORES

VECTORES

AST

ID	Tipo	Dimensiones	Tamaño Total	Ámbito	Estrategia de Almacenamiento	Datos
ud	INT	3	3	Global	Vector 1D: almacenamiento directo lineal Tamaño: 3 elementos Acceso: vector[i] = posición i Datos: [1, 2, 3]	[1,

Se puede observar de mejor manera el AST con alt zoom

