Laboratorio de Organización de Lenguajes y Compiladores 1, Sección P.

PROYECTO OBJ-C

MANUAL TECNICO

FECHA: 27/06/2025

Byron Miguel Galicia Hernandez 201907177
Cesar Armando Garcia Franco 202110378
Alvaro Gabriel Ceballos Gil 202300673
Yulianne Nicole López Elias 202300659

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación, deseñada a través de la interfaz gráfica. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Orientar al usuario en la correcta instalación, configuración y puesta en marcha del sistema, proporcionando una descripción detallada de la estructura de carpetas y archivos, el flujo de inicialización de los componentes (análisis léxico, sintáctico y ejecución), y la interacción entre el backend y el frontend, de modo que se facilite tanto la comprensión del código como la extensión y mantenimiento de sus funcionalidades.

Requisitos Mínimos del Sistema

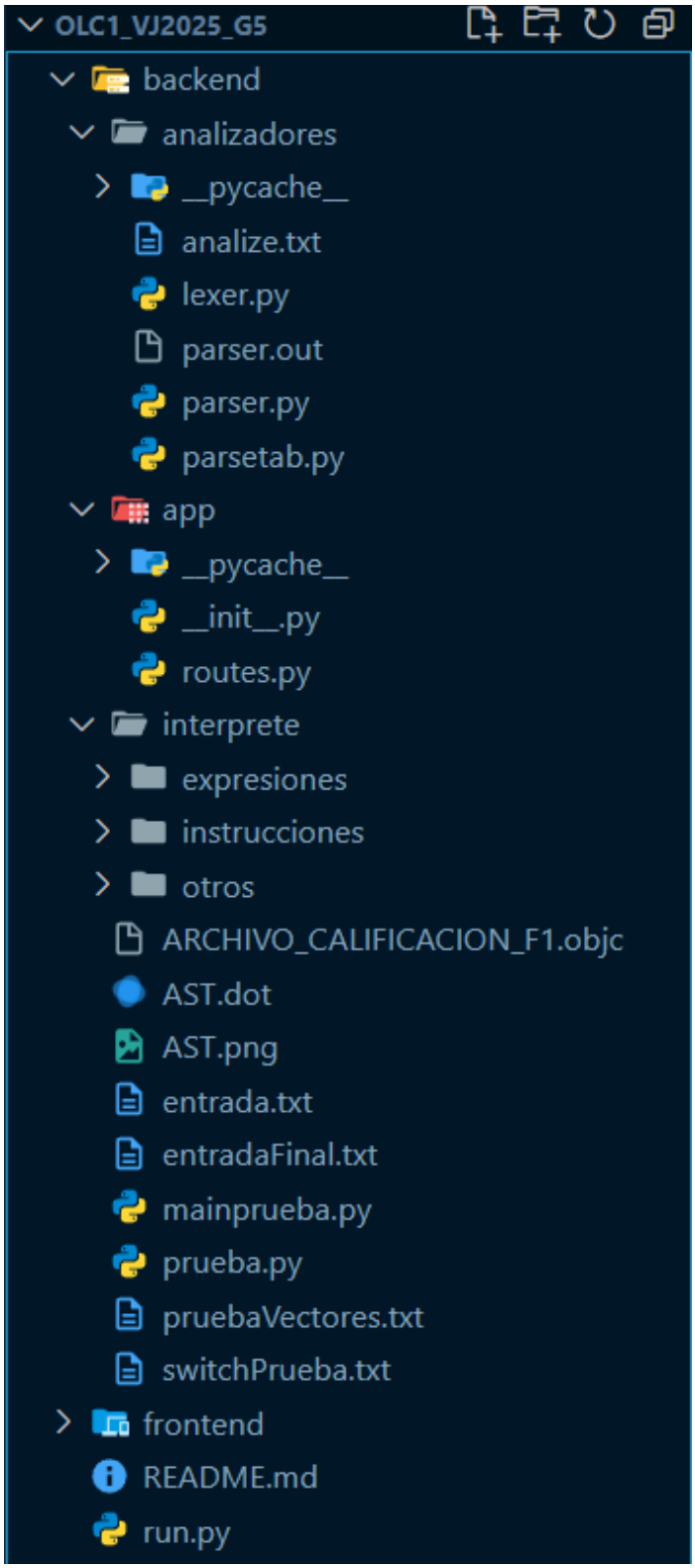
Sistema operativo 64 bits

- Microsoft Windows
- 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)

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



parser.py

Este archivo constituye el núcleo del análisis sintáctico del intérprete, y en él se definen:

Importaciones y configuración inicial

- ply.yacc para construir el analizador sintáctico.
- El lexer (lexer.tokens).
- Todas las clases de instrucciones (Print, Declaracion, Asignacion, estructuras de control, vectores, procedimientos, etc.).
- Todas las clases de expresiones (Literal, Acceso, Aritmetica, Relacional, Logica, funciones nativas Seno, Coseno, Inv, y funciones de vectores Shuffle, Sort).
- Utilidades de entorno (Enviroment), tipos (TipoDato, TipoAritmetica, TipoRelacional, TipoChars) y manejo de errores (TablaErrores, Error).

```python
# Define la función find_column
def find_column(input_text, token):
    last_cr = input_text.rfind('\n', 0, token.lexpos) + 1
    return (token.lexpos - last_cr) + 1

def getTextVal(instrucciones):
    text_var = ''
    for instruccion in instrucciones:
        text_var += instruccion.text_val
    return text_var

def tipoToStr(tipo):
    if tipo == TipoDato.INT:
        return 'int'
    elif tipo == TipoDato.FLOAT:
        return 'float'
    elif tipo == TipoDato.STR:
        return 'str'
    #Cambiar por BOOL
    elif tipo == TipoDato.BOOL:
        return 'bool'
    elif tipo == TipoDato.CHAR:
        return 'char'
    elif isinstance(tipo, TipoChars):
        return tipo.text_val
```

Funciones auxiliaries

- find_column: calcula la columna de un token para reporting de errores.
- getTextVal: concatena los textos de varias instrucciones.
- tipoToStr: convierte un valor de TipoDato (o TipoChars) a su representación en cadena.

Tabla de precedencia

```
# precedencia de operadores
precedence = (
    ('right', 'UMENOS'),

    ('left', 'AND'),
    ('left', 'OR'),
    ('left', 'XOR'),
    ('right', 'NOT'),
    ('left', 'IGUALACION', 'MENOR', 'MAYOR', 'MENOR_IGUAL', 'MAYOR_IGUAL', 'DIFERENCIACION'),
    ('left', 'SUMA', 'RESTA'),
    ('left', 'MULTIPLICACION', 'DIVISION'),
    ('nonassoc', 'POTENCIA', 'MODULO'),
```

precedence: especifica asociatividad y precedencia de operadores (unarios, lógicos, relacionales, aritméticos y de potencia/módulo).

Producciones de la gramática

Entrada y listas

```
# Definición de la gramática
def p_inicio(t):
    '''ini : instrucciones_globales'''
    t[0] = t[1] if t[1] is not None else []
    print('Entrada correcta')

def p_instrucciones_globales(t):
    '''
    instrucciones_globales : instrucciones_globales instruccion_global
                           | instruccion_global
    '''
    if len(t) == 3:
        t[0] = t[1] + [t[2]]
    else:
        t[0] = [t[1]]

def p_instruccion_global(t):
    '''
    instruccion_global : declaracion_procedimiento
                       | instruccion_local
    '''
    t[0] = t[1]

def p_instrucciones_locales(t):
    '''
    instrucciones : instrucciones instruccion_local
                  | instruccion_local
    '''
    if len(t) == 3:
        t[0] = t[1] + [t[2]]
    else:
        t[0] = [t[1]]
```

- p_inicio: reconoce el punto de entrada ini : instrucciones_globales.
- p_instrucciones_globales / p_instruccion_global: distinguen declaraciones de procedimientos de instrucciones locales.
- p_instrucciones_locales / p_instruccion_local: agrupan todas las sentencias que terminan en ; o son bloques/llamadas.

Declaración y acceso de vectores

- p_declaracion_vector / p_declaracion_vector_con_funcion: definen vectores con dimensión y opcional inicialización o función (sort/shuffle).
- p_lista_dimensiones, p_lista_valores_iniciales, p_lista_expresiones: manejan listas de tamaños y valores.
- p_asignacion_vector, p_lista_indices, p_acceso_vector: permiten asignar y leer elementos desde índices múltiples.

Procedimientos

- p_declaracion_procedimiento: define bloques PROC id(params) { ... }.
- p_lista_parametros: lista tipo:id para la cabecera.
- p_llamada_procedimiento, p_lista_argumentos, p_argumento_simple: manejan invocaciones EXEC id(args);.

Instrucciones básicas

- p_instruccion_println, p_declaracion_variable, p_asignacion_variable, p_incremento, p_decremento.

Estructuras de control

- Condicionales: p_instruccion_if, p_base_if.
- Bucles: p_instruccion_while, p_instruccion_for, p_instruccion_dowhile.
- p_instruccion_switch, p_case_unico, p_default_opcional.
- p_estructura_control los agrupa.

Expresiones

- Literales: p_entero, p_decimal, p_cadena, p_caracter, p_literal_booleano.
- Acceso: p_id.
- Aritmética: p_expresion_aritmetica, p_expresion_umenos.
- Relacional: p_relacional.
- Lógica: p_expresion_logica_binaria, p_expresion_not.
- Nativas/vectores: p_expresion_nativa (SENO, COSENO, INV, SHUFFLE, SORT).

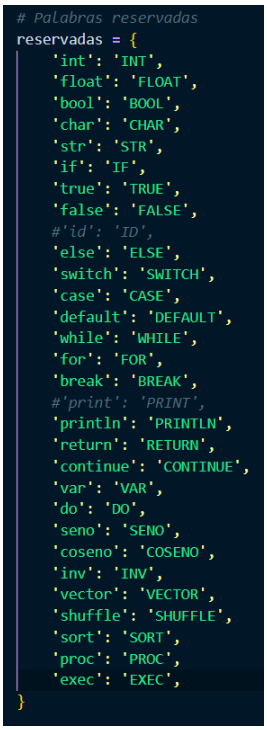
Manejo de errores

- Producciones de recuperación (p_instruccion_error_print, p_instruccion_error_estructura, p_instrucciones_error, p_expresion_error).
- p_error: captura cualquier token inesperado, registra el error y avanza.

lexer.py

El lexer es el componente encargado de transformar la secuencia de caracteres de entrada en una serie de *tokens* que el parser consumirá. A grandes rasgos, este módulo hace lo siguiente:

Definición de palabras reservadas



```
# Palabras reservadas
reservadas = {
    'int': 'INT',
    'float': 'FLOAT',
    'bool': 'BOOL',
    'char': 'CHAR',
    'str': 'STR',
    'if': 'IF',
    'true': 'TRUE',
    'false': 'FALSE',
    #'id': 'ID',
    'else': 'ELSE',
    'switch': 'SWITCH',
    'case': 'CASE',
    'default': 'DEFAULT',
    'while': 'WHILE',
    'for': 'FOR',
    'break': 'BREAK',
    #'print': 'PRINT',
    'println': 'PRINTLN',
    'return': 'RETURN',
    'continue': 'CONTINUE',
    'var': 'VAR',
    'do': 'DO',
    'seno': 'SENO',
    'coseno': 'COSENO',
    'inv': 'INV',
    'vector': 'VECTOR',
    'shuffle': 'SHUFFLE',
    'sort': 'SORT',
    'proc': 'PROC',
    'exec': 'EXEC',
}
```

Se declara un diccionario reservadas que asocia cada palabra clave del lenguaje (tipos,

estructuras de control, funciones nativas, vectores, procedimientos, etc.) con su nombre de token. Al reconocer una cadena que coincide con una palabra reservada, se emitirá el token correspondiente en lugar de un identificador genérico.

Listado de tokens

```
# Lista de tokens
tokens = [
    'ID',
    'ENTERO',
    'SUMA',
    'RESTA',
    'MULTIPLICACION',
    'MODULO',
    'POTENCIA',
    'DIVISION',
    'DECIMAL',
    'CADENAS',
    'CARACTER',
    'COMENTARIO_UNA_LINEA',
    'COMENTARIO_MULTILINEA',
    'IGUALACION',
    'DIFERENCIACION',
    'MENOR',
    'MENOR_IGUAL',
    'MAYOR',
    'MAYOR_IGUAL',
    'IDENTIFICADOR',
    'IGUAL',
    'OR',
    'AND',
    'XOR',
    'NOT',
    'PARA',
    'PARC',
    'PYC',
    'DOS_PUNTOS',
    'ARROBA',
    'COMA',
    'LLA',
    'LLC',
    'UMENOS',
    'INCREMENTO',
    'DECREMENTO',
    'CORCHETE_ABRE',
    'CORCHETE_CIERRA'
] + list(reservadas.values())
```

La lista tokens incluye:

- Tipos básicos de datos (ENTERO, DECIMAL, CADENAS, CARACTER, TRUE/FALSE…).
- Operadores aritméticos (SUMA, RESTA, MULTIPLICACION, DIVISION, MODULO, POTENCIA, INCREMENTO, DECREMENTO).
- Operadores lógicos y relacionales (AND, OR, XOR, NOT, IGUALACION, DIFERENCIACION, MENOR, MAYOR, MENOR_IGUAL, MAYOR_IGUAL).

- Símbolos de agrupación y puntuación (PARA/PARC para paréntesis, LLA/LLC para llaves, CORCHETE_ABRE/CIERRA para corchetes, PYC para punto y coma, COMA, DOS_PUNTOS, etc.).
- Identificadores genéricos ID para nombres de variables, vectores y procedimientos.

Reglas de expresión regular

```
# Expresiones regulares para tokens simples
t_SUMA = r'\+'
t_RESTA = r'-'
t_POTENCIA = r'\*\*'
t_MULTIPLICACION = r'\*'
t_DIVISION = r'/'
t_MODULO = r'%'
t_IGUALACION = r'=='
t_DIFERENCIACION = r'!='
t_MENOR = r'<'
t_MENOR_IGUAL = r'<='
t_MAYOR = r'>'
t_MAYOR_IGUAL = r'>='
t_IGUAL = r'='
t_OR = r'\|\|'
t_AND = r'&&'
t_XOR = r'\^'
t_NOT = r'!'
t_PARA = r'\('
t_PARC = r'\)'
t_PYC = r';'
t_DOS_PUNTOS = r':'
t_ARROBA = r'@'
t_COMA = r','
t_LLA = r'\{'
t_LLC = r'\}'
t_INCREMENTO = r'\+\+'
t_DECREMENTO = r'--'
t_CORCHETE_ABRE = r'\['
t_CORCHETE_CIERRA = r'\]'
```

- Se asocian literales de un solo carácter (por ejemplo, t_SUMA = r'\+') a cada token de símbolos simples.
- Para cadenas más complejas, se definen funciones t_ENTERO, t_DECIMAL, t_CADENAS, t_CARACTER que validan y convierten su valor al tipo Python adecuado (int, float, str).
- t_ID reconoce cualquier identificador (letra o guión bajo seguido de letras, dígitos o guiones bajos), convierte su valor a minúsculas y lo reclasifica como palabra reservada si aparece en el diccionario reservadas.

Comentarios y espacios en blanco

```python
# Expresiones regulares para comentarios
def t_COMENTARIO_UNA_LINEA(t):
    r'//.*'
    pass

def t_COMENTARIO_MULTILINEA(t):
    r'[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]'
    t.value = t.value[2:-2].replace('\n', ' ')  # Eliminar los delimitadores de comentario
    pass
```

- t_COMENTARIO_UNA_LINEA y t_COMENTARIO_MULTILINEA descartan el texto comentado (no generan tokens).
- Los espacios y tabulaciones se ignoran con t_ignore = ' \t'.
- Saltos de línea (t_newline) actualizan el contador de líneas para la ubicación de errores.

Manejo de errores

```python
# Manejo de errores
def t_error(t):
    # Agregando a la tabla de erorres
    err = Error(tipo='Léxico', linea=t.lexer.lineno, columna=find_column(t.lexer.lexdata, t), descripcion=f'Caracter no reconocido: {
    TablaErrores.addError(err)
    t.lexer.skip(1)
```

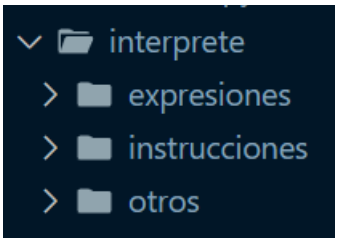
La función t_error captura cualquier carácter que no encaje en las reglas anteriores, crea un objeto Error(tipo='Léxico', ...), lo registra en TablaErrores y avanza un carácter para continuar el análisis en lugar de detenerse.

Analizadores y rutas

- **lexer.py:** Definición de tokens y reglas léxicas con PLY.
- **parser.py:** Gramática principal: instrucciones locales y globales, procedimientos.
- **routes.py:** Define endpoints REST para /datas y /ast.
- **mainprueba.py:** Script principal Flask que inicia el servidor registra rutas.

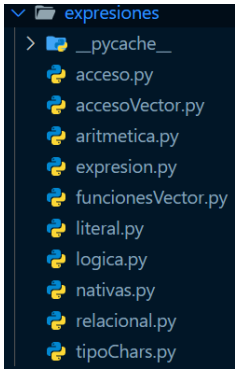
interprete

Dentro de la carpeta interprete reside toda la lógica de ejecución de nuestro lenguaje:



- **expresiones/** contiene las clases que representan y evalúan las distintas expresiones (aritméticas, lógicas, relacionales, literales, nativas, accesos a variables y vectores, etc.).
- **instrucciones/** agrupa los nodos de control de flujo y las acciones del programa (declaraciones, asignaciones, bucles, condicionales, impresión, manejo de vectores, procedimientos, break/continue, etc.), cada uno encargándose de traducir su AST en comportamientos concretos en tiempo de ejecución.
- **otros/** incluye las utilidades y componentes transversales: definición de tipos de dato, manejo de entornos y tablas de símbolos (variables y vectores), reporte de errores, generación de AST, consola de salida y demás ayudas que hacen posible la interpretación coherente y el control de errores de los programas.

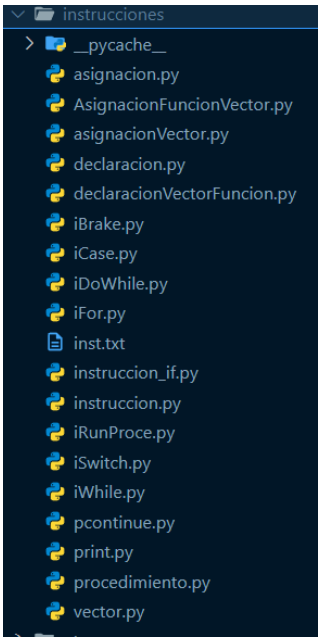
Carpeta expresiones



- **acceso.py:** Acceso a variables simples.
- **accesoVector.py:** Acceso a posiciones de vectores multidimensionales.
- **aritmetica.py:** Implementa operaciones aritméticas y su tipo.
- **literal.py:** Representa literales (enteros, decimales, cadenas, caracteres).
- **logica.py:** Implementa operaciones lógicas y not.

- **nativas.py:** Funciones nativas: seno, coseno, inverso.
- **relacional.py:** Operadores relacionales (==, !=, <, <=, >, >=).
- **tipoChars.py:** Tipo especial STR con tamaño variable.
- **funcionesVector.py:** Funciones sobre vectores: sort, shuffle.
- **expresion.py:** Clase base para todas las expresiones y su evaluación en tiempo de ejecución.

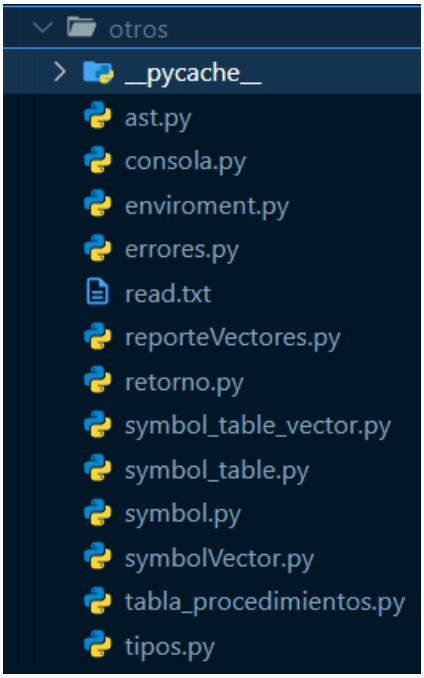
Carpeta instrucciones



- **instruccion.py:** Clase base para todas las instrucciones ejecutables.
- **print.py:** Implementa la instrucción println.
- **declaracion.py:** Declaración de variables y vectores.
- **asignacion.py:** Asignación de variables simples.
- **vector.py:** Declaración de vectores sin inicialización.
- asignacionVector.py: Asignación a posiciones específicas de vectores.
- **AsignacionFuncionVector.py:** Asignación usando resultados de funciones de vectores.
- **declaracionVectorFuncion.py:** Declaración de vectores con inicialización por función.
- **iWhile.py:** Implementa el bucle while.
- **iDoWhile.py:** Implementa el bucle do-while.
- **iFor.py:** Implementa el bucle for.

- **instruccion_if.py:** Implementa if, else-if y else.
- **iSwitch.py:** Implementa switch-case y default.
- **iCase.py:** Caso individual dentro de switch.
- **iBrake.py:** Implementa break para bucles.
- **pcontinue.py:** Implementa continue para bucles.
- **procedimiento.py:** Declaración de procedimientos.
- **RunProce.py:** Ejecución de procedimientos.

Carpeta otros



- **ast.py:** Define las clases y nodos para la generación y recorrido del AST.
- **consola.py:** Módulo para recolectar y exponer la salida de la consola.
- **enviroment.py:** Define el entorno (scope) para variables, vectores y procedimientos.
- **errores.py:** Implementa la estructura de errores y tabla global de errores.
- **reporteVectores.py:** Genera reportes específicos de vectores.
- **retorno.py:** Define la estructura de retorno de expresiones (valor y tipo).
- **tipos.py:** Enumera los tipos de datos soportados (INT, FLOAT, BOOL, CHAR, STR).
- **symbol.py:** Define la estructura base de un símbolo.
- **symbol_table.py:** Tabla de símbolos para variables y procedimientos.
- **symbol_table_vector.py**: Tabla de símbolos especializada para vectores.

- **symbolVector.py:** Estructura de símbolo para vectores.

Gramatica

En dicho proyecto se implemento una gramatica la cual fuera capaz de reconocer las instrucciones del lenguaje de programacion, a continuacion se muestra la gramatica utilizada en el proyecto para la fase 1 y 2.

<ini> ::= <instrucciones_globales>

<instrucciones_globales> ::= <instrucciones_globales> <instruccion_global>
| <instruccion_global>

<instruccion_global> ::= <declaracion_procedimiento>
| <instruccion_local>

<instrucciones> ::= <instrucciones> <instruccion_local>
| <instruccion_local>
| ε

<instruccion_local> ::= <instruccion_println>
| <declaracion_variable> ";"
| <asignacion_variable> ";"
| <estructura_control>
| <incremento> ";"
| <decremento> ";"
| <declaracion_vector> ";"
| <llamada_procedimiento>

<declaracion_vector> ::= "VECTOR" "[" <tipo> "]" ID "(" <lista_dimensiones> ")"
| "VECTOR" "[" <tipo> "]" ID "(" <lista_dimensiones> ")" "="
<lista_valores_iniciales>
| "VECTOR" "[" <tipo> "]" ID "(" <lista_dimensiones> ")" "=" "SORT" "("
<expresion> ")"
| "VECTOR" "[" <tipo> "]" ID "(" <lista_dimensiones> ")" "=" "SHUFFLE" "("
<expresion> ")"

<lista_dimensiones> ::= <lista_dimensiones> "," <expresion>
| <expresion>

<lista_valores_iniciales> ::= <lista_valores_iniciales> "," "[" <lista_expresiones> "]"
| "[" <lista_expresiones> "]"

<lista_expresiones> ::= <lista_expresiones> "," <expresion>
| <expresion>

<asignacion_variable> ::= ID <lista_indices> "=" <expresion>

<lista_indices> ::= <lista_indices> "[" <expresion> "]"
| "[" <expresion> "]"

<literal> ::= ID <lista_indices>

<declaracion_procedimiento> ::= "PROC" ID "(" <lista_parametros> ")" "{"
<instrucciones> "}"

<lista_parametros> ::= <lista_parametros> "," <tipo> ":" ID
| <tipo> ":" ID
| ε

<llamada_procedimiento> ::= "EXEC" ID "(" <lista_argumentos> ")" ";"

<lista_argumentos> ::= <lista_argumentos> "," <argumento_simple>
| <argumento_simple>
| ε

<argumento_simple> ::= ID
| ENTERO
| DECIMAL
| CARACTER
| CADENAS
| "TRUE"
| "FALSE"

<estructura_control> ::= <instruccion_while>
| <instruccion_if>
| <instruccion_switch>

| <instruccion_dowhile>
| <instruccion_for>

<instruccion_while> ::= "WHILE" "(" <expresion> ")" "{" <instrucciones> "}"

<instruccion_for> ::= "FOR" "(" <declaracion_variable> ";" <expresion> ";"
<actualizacion> ")" "{" <instrucciones> "}"

<actualizacion> ::= <incremento>
| <decremento>
| <expresion>
| <asignacion_variable>

<instruccion_switch> ::= "SWITCH" "(" <expresion> ")" "{" <lista_case>
<default_opcional> "}"

<instruccion_dowhile> ::= "DO" "{" <instrucciones> "}" "WHILE" "(" <expresion> ")" ";"

<lista_case> ::= <lista_case> <case_unico>
| <case_unico>

<case_unico> ::= "CASE" <expresion> ":" <instrucciones>

<default_opcional> ::= "DEFAULT" ":" <instrucciones>
| ε

<instruccion_if> ::= <base_if>
| <base_if> "ELSE" "{" <instrucciones> "}"
| <base_if> "ELSE" <instruccion_if>

<base_if> ::= "IF" "(" <expresion> ")" "{" <instrucciones> "}"

<incremento> ::= ID "++"

<decremento> ::= ID "—"

<instruccion_println> ::= "PRINTLN" "(" <expresion> ")" ";"

<declaracion_variable> ::= <tipo> ID "=" <expresion>
| <tipo> ID

<asignacion_variable> ::= ID "=" <expresion>

<expresion> ::= <aritmetica>
| <literal>
| <relacional>
| "(" <expresion> ")"
| <expresion> "OR" <expresion>
| <expresion> "AND" <expresion>
| <expresion> "XOR" <expresion>
| "-" <expresion>
| "!" <expresion>
| "SENO" "(" <expresion> ")"
| "COSENO" "(" <expresion> ")"
| "INV" "(" <expresion> ")"
| "SHUFFLE" "(" <expresion> ")"
| "SORT" "(" <expresion> ")"

<aritmetica> ::= <expresion> "+" <expresion>
| <expresion> "-" <expresion>
| <expresion> "*" <expresion>
| <expresion> "/" <expresion>
| <expresion> "**" <expresion>
| <expresion> "%" <expresion>

<relacional> ::= <expresion> "==" <expresion>
| <expresion> "!=" <expresion>
| <expresion> "<" <expresion>
| <expresion> ">" <expresion>

| <expresion> "<=" <expresion>

| <expresion> ">=" <expresion>

<literal> ::= ENTERO

| DECIMAL

| CADENAS

| CARACTER

| ID

| "TRUE"

| "FALSE"

<tipo> ::= "int"

| "float"

| "str"

| "char"

| "bool"

<instruccion> ::= "BREAK" ";"

| "CONTINUE" ";"