TECNOLÓGICO
DE MONTERREY®

Proyecto CompilarAD23

Yannin Flores Maza

A01324745

25/11/2023

Índice:

1. DOCUMENTACIÓN DEL PROYECTO

1.1. DESCRIPCIÓN DEL PROYECTO

1.1.1. Propósito y Alcance, Análisis de Requerimientos y Procesos de Desarrollo.

- Propósito y Alcance.

Este proyecto deberá abarcar diversas facetas del proceso de compilación, ofreciendo una visión completa y detallada de cómo un programa escrito en un lenguaje de alto nivel es transformado en una forma que puede ser ejecutada por una máquina y mediante el uso del lenguaje Python en conjunto su indispensable herramienta PLY (Python Lex-Yacc) para su análisis léxico y sintáctico.

Como objetivo inicial del proyecto CompilarAD23 es crear un compilador que no solo traduzca el código fuente a código de máquina, sino que también maneje eficientemente todos los aspectos intermedios del proceso de compilación. Estos componentes se distribuyen en distintas fases:

El proyecto CompilarAD23 no solo busca implementar las funcionalidades estándar de un compilador sino también introducir innovaciones en términos de eficiencia, manejo de errores y facilidad de uso. Al proporcionar una vista completa del ciclo de vida del código, desde su creación hasta su ejecución, este proyecto sirve como un recurso educativo invaluable, permitiendo a los usuarios experimentar y aprender sobre la teoría y práctica de la compilación de lenguajes de programación.

- Análisis de Requerimientos.

El proyecto CompilarAD23 presenta una visión ambiciosa que abarca diversas facetas del proceso de compilación. El análisis de requerimientos se puede desglosar en los siguientes puntos:

- Comprensión Integral del Proceso de Compilación:

El proyecto busca ofrecer una visión completa y detallada del proceso de compilación, desde el código fuente en un lenguaje de alto nivel hasta la ejecución en una máquina. Este enfoque integral sugiere una necesidad de abordar cada fase del proceso de manera exhaustiva.
Uso de Python y PLY (Python Lex-Yacc):

La elección de utilizar Python y la herramienta PLY para el análisis léxico y sintáctico implica una decisión específica de tecnología. Este requisito establece la base tecnológica para el desarrollo del compilador.

- Manejo Eficiente de Aspectos Intermedios:

El objetivo principal es desarrollar un compilador que no solo traduzca a código de máquina, sino que también gestione eficientemente los aspectos intermedios del proceso de compilación. Esto sugiere un enfoque en la calidad y eficiencia de las representaciones intermedias.

- Innovación en Eficiencia, Manejo de Errores y Facilidad de Uso:

El proyecto no solo busca implementar funcionalidades estándar, sino también introducir innovaciones en términos de eficiencia, manejo de errores y facilidad de uso. Este requisito destaca la importancia de la mejora continua y la usabilidad del compilador.

Procesos de Desarrollo.

El desarrollo del proyecto CompilarAD23 se divide en distintas fases, cada una destinada a abordar aspectos específicos del proceso de compilación:

- Análisis Léxico:

La primera fase implica descomponer el código fuente en tokens, estableciendo la base para la comprensión de la estructura básica del programa.

- Análisis Sintáctico:

En esta fase, se construye un árbol sintáctico que representa la estructura gramatical del programa, permitiendo entender cómo se organizan los tokens en construcciones más complejas.

- Análisis Semántico:

El compilador verifica la coherencia de tipos y otras restricciones semánticas, asegurando que el programa sea gramaticalmente correcto y lógicamente coherente.

- Generación de Código Intermedio:

Transformación de estructuras de alto nivel en representaciones intermedias, como cuádruplos, para facilitar la traducción final a código de máquina.

- Optimización de Código:

Se busca mejorar la eficiencia del código intermedio en términos de tiempo de ejecución y uso de memoria.

- Generación de Código Ejecutable:

La etapa final traduce el código intermedio en código de máquina o en un formato interpretable por una máquina virtual.

El proyecto se destaca no solo por implementar funcionalidades estándar sino por introducir innovaciones, proporcionando una visión completa del ciclo de vida del código. Este enfoque lo posiciona como un recurso educativo valioso, permitiendo a los usuarios experimentar y aprender sobre la teoría y práctica de la compilación de lenguajes de programación.

1.1.2. Descripción del Lenguaje.

- Características Principales.

Declaración de Variables:
El proyecto CompilarAD23 permite declarar variables tanto locales como globales. La declaración se realiza especificando el tipo de dato seguido del nombre de la variable.
Estructuras de Control:

Ciclos:
Se manejan estructuras de ciclo como while y for, que permiten la ejecución repetida de bloques de código basados en condiciones booleanas.
Condicionales: Incluye estatutos if, else if y else, facilitando la ejecución condicional de diferentes bloques de código.

Funciones:
CompilarAD23 soporta la declaración de funciones tanto con valor de retorno como funciones tipo void. Las funciones deben ser declaradas antes del bloque principal y se invocan mediante el uso de paréntesis con los parámetros correspondientes.

Arreglos:
El lenguaje soporta la declaración y manipulación de arreglos unidimensionales.

Expresiones:
Permite realizar expresiones aritméticas, lógicas y de comparación, abarcando una amplia gama de operaciones matemáticas y lógicas.

Tipos de Datos:
Soporta tipos de datos como enteros, flotantes, caracteres y booleanos, permitiendo una amplia flexibilidad en la manipulación de datos.

Manejo de Errores:
Incluye un sistema de manejo de errores que detecta y reporta problemas como tipos de datos incompatibles, variables no declaradas, problemas de sintaxis, entre otros.

Generación de Cuádruplos:
Implementa un sistema de generación de cuádruplos para la representación intermedia de código, facilitando la ejecución y optimización del código.

Memoria Virtual y Gestión de Recursos:
Utiliza un sistema de memoria virtual para la asignación y manejo eficiente de variables y constantes, así como para la gestión de memoria en tiempo de ejecución.

Léxico y Análisis Sintáctico:
Emplea herramientas como PLY para el análisis léxico y sintáctico, asegurando que el código fuente cumpla con las reglas gramaticales del lenguaje.

Funciones Especiales y Operaciones Matemáticas:
Incluye funciones para operaciones matemáticas avanzadas y manejo de estructuras de datos complejas, expandiendo las capacidades del lenguaje más allá de la programación básica.

- Errores de Compilación y Ejecución.

- **Error, tipo de dato inesperado en operación:**

Indica que se intentó realizar una operación con tipos de datos incompatibles o inesperados.

- **Error, tamaño de arreglo inválido:**

Señala que se ha declarado un arreglo con un tamaño no válido o que se intentó acceder a un índice fuera de los límites del arreglo.

- **Error, uso incorrecto de operadores lógicos**

Indica que se utilizó un operador lógico (como AND, OR) en un contexto no permitido o con tipos de datos incompatibles.

- **Error, división por cero:**

Ocurre cuando hay un intento de dividir un número por cero durante la ejecución.

- **Error, tipo de retorno de función no coincide:**

Indica que el tipo de valor que retorna una función no coincide con el tipo declarado.

- **Error, uso de variable no inicializada:**

Señala que se intenta utilizar una variable que no ha sido inicializada previamente.

- **Error, llamada a función con número incorrecto de argumentos:**

Indica que el número de argumentos en la llamada a una función no coincide con el número de parámetros declarados en su definición.

- **Error, uso incorrecto de constantes:**

Ocurre cuando se intenta modificar el valor de una constante o se utiliza de manera inadecuada.

- **Error, desbordamiento de memoria:**

Indica que se ha excedido el límite de memoria asignado para variables o estructuras de datos.

- **Error, operación ilegal en cuádruplo:**

Señala que se intentó ejecutar una operación en la máquina virtual que no está permitida o no está definida en los cuádruplos.

- **Error, sintaxis incorrecta en declaración de variables o funciones:**

Indica un error de sintaxis al declarar variables o funciones, como nombres incorrectos o falta de símbolos requeridos.

- **Error, referencia a identificador desconocido:**

Ocurre cuando se hace referencia a una variable o función que no ha sido declarada.

- **Error, sobrecarga de función no permitida:**

Indica que se intentó declarar dos funciones con el mismo nombre y parámetros, lo cual no está permitido.

- **Error, incompatibilidad de tipos en asignación:**

Señala que se intenta asignar un valor a una variable de un tipo diferente al que puede almacenar.

- **Error, uso indebido de estructuras de control:**

Indica que las estructuras de control (como IF, WHILE, FOR) se usaron de manera incorrecta o con condiciones inadecuadas.

1.1.3. Descripción del Compilador.

- Herramientas utilizadas.

Para el desarrollo de este proyecto se llegó a involucrar distintas herramientas como en el primer caso, PLY, el cual representa una implementación pura de Python de las populares herramientas de procesamiento de lenguajes Lex y Yacc. En CompilarAD23, PLY se usa para crear el analizador léxico y el analizador sintáctico que son fundamentales en el proceso de compilación, permitiendo la conversión del código fuente en tokens y luego en una estructura de datos que representa la sintaxis del programa.

También el módulo sys, que forma parte de la biblioteca estándar de Python y proporciona acceso a algunas variables y funciones que interactúan fuertemente con el intérprete de Python. En el contexto del compilador, sys puede manejar argumentos de línea de comandos, ya sea para terminar el programa si se encuentra un error crítico durante la compilación.

genericpath y os.path (isfile): Estos módulos están relacionados con las operaciones del sistema de archivos. Se usan para verificar si un archivo existe en el sistema de archivos, lo cual es esencial para leer el código fuente del programa que debe ser compilado.

matplotlib.pyplot: Este es un módulo de Matplotlib, una biblioteca de trazado para Python que proporciona funciones similares a las de MATLAB. matplotlib.pyplot se utiliza para crear gráficos y visualizaciones. En el contexto del compilador, fue de gran utilidad para visualizar datos resultantes de la ejecución del programa compilado, como gráficos de resultados de cálculos o para mostrar gráficos de depuración.

NumPy es una biblioteca esencial para la computación científica en Python, el cual ofrece un objeto de matriz multidimensional y herramientas para trabajar con estas matrices. Es ampliamente utilizada para operaciones matemáticas de alto rendimiento y podría ser empleada en CompilarAD23 para realizar cálculos numéricos complejos o manejar grandes conjuntos de datos dentro del lenguaje que se está compilando.

random: Este módulo se implementó para generar números pseudoaleatorios. La función random() devuelve un flotante aleatorio en el rango [0.0, 1.0). En el compilador, esta función puede se llevó a cabo para funcionalidades que requieran aleatoriedad, como la generación de datos de prueba o la implementación de funciones aleatorias.

statistics: Este módulo proporciono funciones para calcular estadísticas matemáticas de datos numéricos. Las funciones como mean(), mode() y variance() se usaron para calcular el promedio, la moda y la varianza de una secuencia de números,

respectivamente con capacidades para realizar análisis estadísticos.

- Análisis Léxico y Sintaxis.

Token	Palabras Reservadas	Descripción
PROGRAM	program	Palabra reservada para inicio de programa
VARS	vars	Palabra reservada para declaración de variables
FUNCTION	function	Palabra reservada para definición de funciones
MAIN	main	Palabra reservada para la función principal
RETURN	return	Palabra reservada para retorno de valores
READ	read	Palabra reservada para lectura de datos
WRITE	write	Palabra reservada para escritura de datos
IF	if	Palabra reservada para condicionales
THEN	then	Palabra reservada para bloque THEN en condicionales
ELSE	else	Palabra reservada para bloque ELSE en condicionales
WHILE	while	Palabra reservada para bucles WHILE
DO	do	Palabra reservada para iniciar bloques de bucles
FOR	for	Palabra reservada para bucles FOR
TO	to	Palabra reservada para rango en bucles FOR
INT	int	Palabra reservada para tipo entero
FLOAT	float	Palabra reservada para tipo flotante
CHAR	char	Palabra reservada para tipo carácter
TRUE	true	Palabra reservada para valor booleano verdadero
FALSE	false	Palabra reservada para valor booleano falso
VOID	void	Palabra reservada para funciones sin retorno
POW	pow	Palabra reservada para función de potencia
RAND	rand	Palabra reservada para función aleatoria
MED	med	Palabra reservada para función de mediana
MODA	moda	Palabra reservada para función de moda
VAR	var	Palabra reservada para función de varianza
REG	reg	Palabra reservada para función de regresión
PLOT	plot	Palabra reservada para función de trazado gráfico
Token	**Expresiones Regulares**	**Descripción**
ID	[a-z][a-zA-Z0-9]*	Identificador
CTE_I	[-]?[0-9]+	Constante entera
CTE_F	[-]?[0-9]+([.][0-9]+)	Constante flotante
CTE_C	`(L)?'([^\\n]	(\.))*?'`

CTE_S	"(.*?)"	Constante de cadena
PLUS	\+	Operador de suma
MINUS	-	Operador de resta
TIMES	*	Operador de multiplicación
DIVIDE	/	Operador de división
MOD	%	Operador de módulo
AND	&	Operador lógico AND
OR	||	Operador lógico OR
EQUALS	==	Operador de igualdad
DIFFERENCE	!=	Operador de desigualdad
GTHAN	>	Operador de mayor que
LTHAN	<	Operador de menor que
GEQUAL	>=	Operador de mayor o igual que
LEQUAL	<=	Operador de menor o igual que
SEMICOLON	;	Punto y coma
COMMA	,	Coma
EQUAL	=	Signo de asignación
LPAREN	\(	Paréntesis izquierdo
RPAREN	\)	Paréntesis derecho
LBRACKET	\[	Corchete izquierdo
RBRACKET	\]	Corchete derecho
LBRACE	\{	Llave izquierda
RBRACE	\}	Llave derecha

Gramática Formal

```
# Crear y agregar cuádruplo que indica el final del programa
    "program : PROGRAM ID init SEMICOLON vars programp main"

# Establecer el ID del programa y añadir cuádruplo inicial
    "init :"

# Definición vacía para posible lista de funciones
    """programp : function programp
    | empty"""

# Procesamiento de listas de variables de un tipo
    """varsp : type varspp SEMICOLON varsp
    | empty"""

# Lista de variables separadas por comas
    """varspp : varsppp
    | varsppp COMMA varspp"""
```

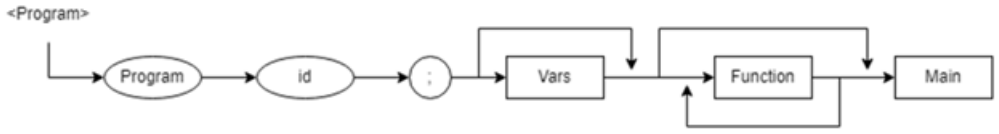
```
# Asignación de un identificador a una variable
    "varsppp : ID varspppp"

# Definición para arreglos o variable simple
    """varspppp : LBRACKET CTE_I RBRACKET
    | empty"""

# Definición de una función y generación de cuádruplo para el final de función
    """function : FUNCTION functionp ID idFuncionActual parameters vars statements
    | empty"""

# Definición del tipo de retorno de una función
    """functionp : type
    | VOID"""

# Establecer el identificador de la función actual
    "idFuncionActual :"

# Definición de los parámetros de una función
    "parameters : LPAREN parametersp RPAREN"

# Lista de parámetros para una función
    """parametersp : type ID parameterArray parameterspp
    | empty"""

# Definición de un parámetro como arreglo o variable simple
    """parameterArray : LBRACKET CTE_I RBRACKET
    | empty"""

# Procesamiento de parámetros adicionales en funciones, separados por comas
    """parameterspp : COMMA parametersp
    | empty"""

# Definición de la función principal 'main'
    "main : MAIN mainID LPAREN RPAREN statements"

# Establecer el identificador de la función principal y completar cuádruplo de salto
    "mainID :"

# Bloque de declaraciones o instrucciones
    "statements : LBRACE statementsp RBRACE"

# Procesamiento de múltiples declaraciones o instrucciones
    """statementsp : statementspp SEMICOLON statementsp
    | statementsppp statementsp
    | empty"""

# Diferentes tipos de declaraciones, incluyendo asignación, llamada a funciones, retorno, lectura y
escritura
    """statementspp : assignment
    | voidCall
    | return
    | read
    | write"""
```

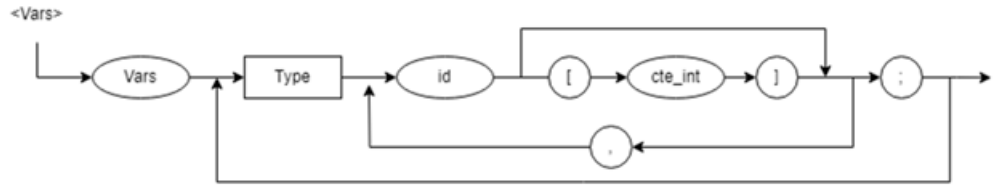
```
# Llamada a funciones sin retorno o funciones especiales
    """voidCall : call
    | specCall"""

# Declaraciones para estructuras de control, como condicionales y bucles
    """statementsppp : condition
    | loop"""

# Procesamiento de una asignación
    "assignment : variable EQUAL assignmentp"

# Asignar una expresión o el resultado de una llamada a función
    """assignmentp : expression
    | funcCall"""

# Llamada a funciones con retorno
    """funcCall : call
    | specCall"""

# Procesamiento de una llamada a función
    "call : ID initParams LPAREN callp RPAREN"

# Procesamiento de llamadas a funciones especiales como 'int', 'float', etc.
    "specCall : specCallp initParams LPAREN callp RPAREN"

# Captura el identificador de la función especial a llamar
    """specCallp : INT
    | FLOAT
    | POW
    | RAND
    | MED
    | MODA
    | VAR
    | REG
    | PLOT"""

# Procesa una expresión en una llamada a función y maneja parámetros adicionales
    """callp : expression callpp
    | empty"""

# Maneja múltiples parámetros en llamadas a funciones, separados por comas
    """callpp : COMMA callp
    | empty"""

# Procesa la instrucción 'return' en funciones
    "return : RETURN LPAREN expression RPAREN"

# Genera cuádruplos para leer entradas del usuario
    "read : READ initParams LPAREN readp RPAREN"

# Contabiliza los parámetros en la instrucción 'read'
    "readp : variable readpp"
```

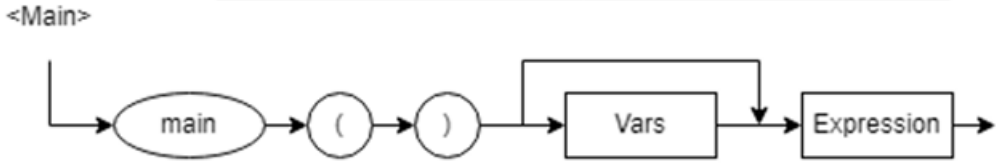
```
# Maneja múltiples variables para leer, separadas por comas
    """readpp : COMMA readp
    | empty"""

# Crea cuádruplos para imprimir todo en la llamada en la misma línea y luego imprimir un salto de línea
    "write : WRITE initParams LPAREN writep RPAREN"

# Inicializa el contador de parámetros
    "initParams :"

# Cuenta la cantidad de parámetros en 'write'
    """writep : expression writepp
    | CTE_S string writepp"""

# Maneja parámetros adicionales para imprimir, separados por comas
    """writepp : COMMA writep
    | empty"""

# Procesa la estructura condicional 'if-then-else'
    "condition : IF LPAREN expression c1 THEN statements conditionp c3"

# Procesa la parte 'else' de la estructura condicional
    """conditionp : c2 ELSE statements
    | empty"""

# Comprueba si la expresión es de tipo booleano y añade un salto a la pila
    "c1 : RPAREN"

# Genera un cuádruplo para saltar al final del 'else' y llena el salto desde la pila
    "c2 :"

# Llena el salto desde la pila
    "c3 :"

# Define bucles 'while' y 'for'
    """loop : while
    | for"""

# Define la estructura del bucle 'while'
    "while : WHILE w1 LPAREN expression w2 DO statements w3"

# Añade un salto a la pila para el bucle 'while'
    "w1 :"

# Comprueba si la expresión del 'while' es de tipo booleano y añade un salto a la pila
    "w2 : RPAREN"

# Genera un cuádruplo para volver al inicio del 'while'
    "w3 :"
```

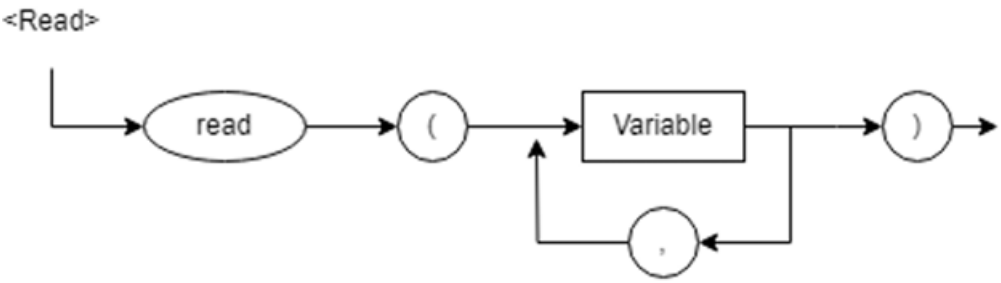
```
# Define la estructura del bucle 'for'
    "for : FOR ID EQUAL expression f1 expression f2 statements"

# Comprueba que el valor inicial del 'for' sea de tipo entero y añade un salto a la pila
    "f1 : TO"

# Genera cuádruplos para probar si la variable es menor o igual al límite y para ir al final del 'for'
cuando sea falso
    "f2 : DO"

# Define expresiones y factores
    """expression : expression expressionp
    | factor"""

# Genera cuádruplos para operaciones binarias
    """expressionp : AND expression
    | OR expression
    | LTHAN expression
    | GTHAN expression
    | EQUALS expression
    | DIFFERENCE expression
    | LEQUAL expression
    | GEQUAL expression
    | PLUS expression
    | MINUS expression
    | TIMES expression
    | DIVIDE expression
    | MOD expression"""

# Procesa factores que pueden ser una expresión entre paréntesis, una variable o una constante
    """factor : LPAREN expression RPAREN
    | var_cte
    | variable"""

# Define una variable con su identificador y procesa posibles índices o propiedades
    "variable : ID variablep"

# Carga una variable a la pila de operandoos, manejando índices de arreglo
    """variablep : LBRACKET expression RBRACKET
    | empty"""

# Define constantes de diferentes tipos
    """var_cte : TRUE bool
    | FALSE bool
    | CTE_C char
    | CTE_S string
    | CTE_I int
    | CTE_F float"""

# Maneja constantes booleanas
    "bool :"
```

```
# Maneja constantes de tipo carácter
    "char :"

# Maneja constantes de tipo cadena
    "string :"

# Maneja constantes de tipo entero
    "int :"

# Maneja constantes de tipo flotante
    "float :"

# Define el tipo de datos para variables o constantes
    """type : INT
    | FLOAT
    | CHAR"""

# Define un nodo vacío en la gramática
    "empty :"

# Maneja errores sintácticos
    "error :"
```

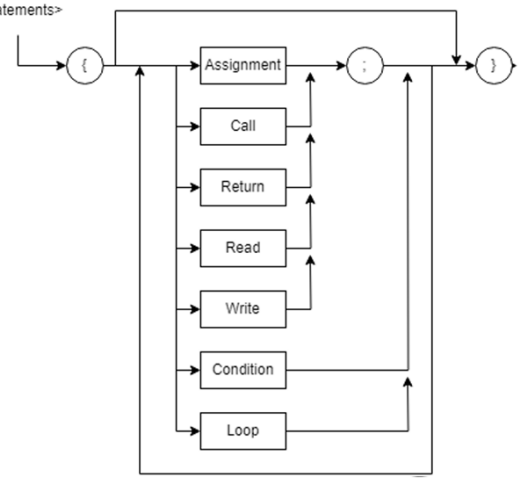




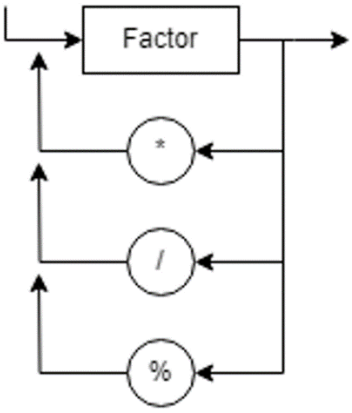




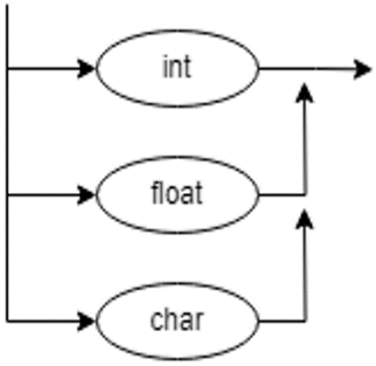
<Statements>



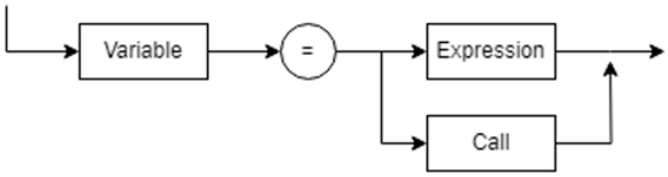
<Term>



<Type>

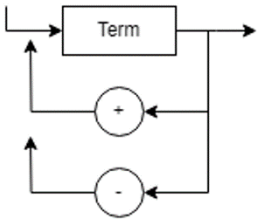


<Assignment>

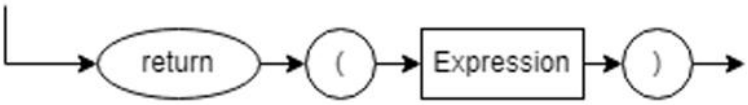


<While>

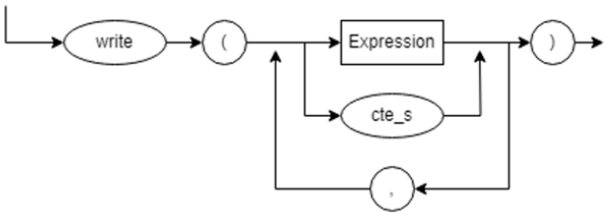
<Arit_Exp>



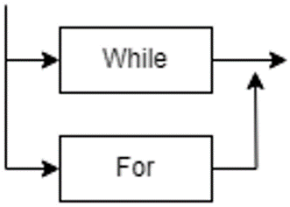
<Return>



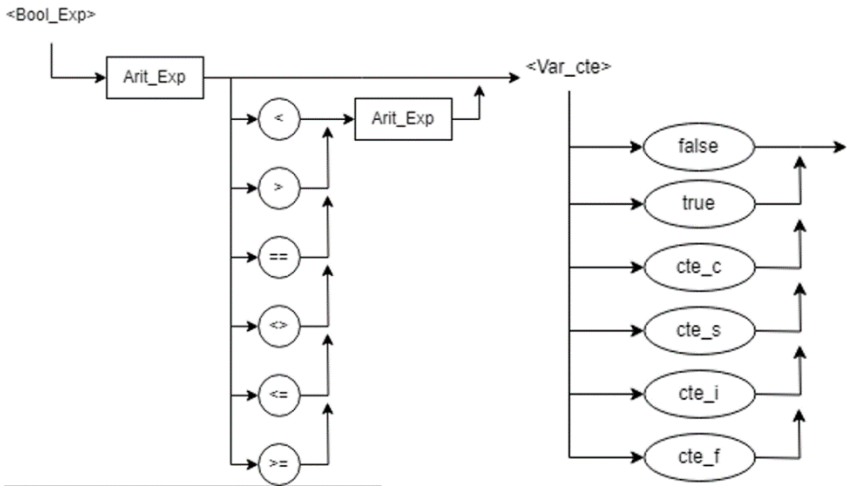
<Write>



<Loop>



<For>



- Generación de Código Intermedio y Análisis Semántico

Declaración de variables y funciones:

```
# En el código del parser, se verifica y añade variables al directorio de funciones.

def checkVarOverlap(id, arrSize):
    global directorioFunciones
    overlap = False
    if id in directorioFunciones[programID]["vars"]:
        overlap = True
    if idFuncionActual != programID:
        if id in directorioFunciones[idFuncionActual]["vars"]:
            overlap = True

    if not overlap:
        var = {"type": tipoDatoActual, "arrSize": arrSize}
        if idFuncionActual == programID:
            dir = MemoriaG.asignarMemoria(var)
        else:
            dir = MemoriaL.asignarMemoria(var)
        var["dir"] = dir

        directorioFunciones[idFuncionActual]["vars"][id] = var
    else:
```

Declaración de Variables: El código maneja la declaración de variables tanto a nivel global como local (dentro de funciones). Se verifica si una variable ya ha sido declarada antes de añadirla al directorio de funciones.

```
# En el código del parser, se verifica y añade funciones al directorio de funciones.
def checkFuncOverlap():
    global directorioFunciones, idFuncionActual
    if idFuncionActual in directorioFunciones:
        print(f"Function name {idFuncionActual} has been declared elsewhere")
        sys.exit()
    else:
        directorioFunciones[idFuncionActual] = {"type": tipoDatoActual, "dir":
len(CuadruploList), "vars": {}}
        if tipoDatoActual != "void":
            id = idFuncionActual
            idFuncionActual = programID
            checkVarOverlap(id, 0)
            idFuncionActual = id
```

Declaración de Funciones: El código también maneja la declaración de funciones. Se verifica si una función ya ha sido declarada antes de añadirla al directorio de funciones.

Generación de acceso a variables dimensionadas:

```
# En el código del parser, se maneja la carga de la variable a la pila de operandos y se procesan posibles índices o propiedades.
def p_variablep(p):
    """variablep : LBRACKET expression RBRACKET
    | empty"""
    findIdType(p[-1])
    if len(p) == 4:
        global CuadruploList

        var = pilaOperadores.pop()
        exp = pilaOperadores.pop()
        if exp.get("type") != "int":
            print(
                f"Expression for array in line {p.lineno(1)!r} needs to result in integer type"
            )
            sys.exit()
        else:
            newCuadruplo = Cuadruplo("VER", exp, EMPTY, var.get("arrSize"))
            CuadruploList.append(newCuadruplo)
            checkConstOverlap({"type": var.get("type"), "id": var.get("dir")})
            pilaOperadores.append(exp)
            genCuadruplo("+")
            resultado = pilaOperadores.pop()
            resultado["dir"] = f"*{resultado.get('dir')}"
            pilaOperadores.append(resultado)
```

Manejo de Índices de Arreglos: El código incluye la generación de cuádruplos para acceder a elementos de arreglos mediante el uso de índices.

Generación de scopes de memoria (gosub):

```
def p_specCall(p):
    "specCall : specCallp initParams LPAREN callp RPAREN"
    global contadorParametros, contadorResultados
    id = p[1]
    p[0] = p[1]
    # Crear y añadir cuádruplo para iniciar la llamada a la función especial
    newCuadruplo = Cuadruplo("ERA", EMPTY, EMPTY, id)
    CuadruploList.append(newCuadruplo)
    # Procesamiento específico según el tipo de función especial
    if id == "int":
        # Verificar número correcto de parámetros para 'int' y sus tipos
        if contadorParametros != 1:
            print(f"Wrong number of parameters in call to int")
            sys.exit()

        parameter = pilaOperadores.pop()
        if parameter.get("type") != "float" or parameter.get("arrSize") != 0:
            print(
                f"Parameter types or arrSize in line {p.lineno!r} do not match call
to {id}"
            )
            sys.exit()
        newCuadruplo = Cuadruplo("PARAM", parameter, EMPTY, 0)
        CuadruploList.append(newCuadruplo)

        genresultado("int")

    elif id == "float":
        # Verificar número correcto de parámetros para 'float' y sus tipos
        if contadorParametros != 1:
            print(f"Wrong number of parameters in call to int")
            sys.exit()

        parameter = pilaOperadores.pop()
        if parameter.get("type") != "int" or parameter.get("arrSize") != 0:
```

Manejo de Scope Local: Se manejan los scopes locales dentro de funciones mediante la asignación de memoria local.

Generación de direcciones virtuales para parser y máquina virtual:

```
# En el código del parser, se utilizan direcciones virtuales para las variables locales y temporales, asignadas por el manejador de memoria correspondiente.

def genresultado(resultadoType):
    global contadorResultados
    resultado = "resultado" + str(contadorResultados)
    contadorResultados += 1
    isLocal = programID != idFuncionActual
    dir = MemoriaT.asignarMemoria(resultadoType, isLocal)
    pilaOperadores.append({"id": resultado, "type": resultadoType, "dir": dir, "arrSize": 0})
```

Manejo de Memoria Virtual: Se utilizan direcciones virtuales para las variables locales y temporales, y estas son asignadas por el manejador de memoria correspondiente (por ejemplo, MemoriaG para variables globales, MemoriaL para variables locales, etc.).

Procesamiento de Cuádruplos y Selección de Operación:

```
def genCuadruplo(operador):
    # Crea cuádruplos para operaciones simples
    global pilaOperadores, CuadruploList, contadorResultados, tipoDatoActual
    operando2 = pilaOperadores.pop()
    operando1 = pilaOperadores.pop()
    resultadoType = cubo[operando1.get("type")][operando2.get("type")][operador]

    if resultadoType != "error":
        genresultado(resultadoType)
        newCuadruplo = Cuadruplo(operador, operando1, operando2, pilaOperadores[-1].get("dir"))
        CuadruploList.append(newCuadruplo)
    else:
        print(
            f"Desajuste de tipo causado por {operador} en {operando1.get('id')} y {operando2.get('id')}"
        )
        sys.exit()
```

Generación de Cuádruplos: El código genera cuádruplos para diversas operaciones, como asignaciones, operaciones aritméticas, llamadas a funciones, etc.

Operando Izquierdo	Operando Derecho	+	−	/	%	<	>	<=	>=	==	<>	&	|
int	int	int	int	int	int	bool	bool	bool	bool	bool	bool	error	error
int	float	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error
int	char	error	error	error	error	error	error	error	error	error	error	error	error
int	string	error	error	error	error	error	error	error	error	error	error	error	error
int	bool	error	error	error	error	error	error	error	error	error	error	error	error
float	int	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error
float	float	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error
float	char	error	error	error	error	error	error	error	error	error	error	error	error
float	string	error	error	error	error	error	error	error	error	error	error	error	error
float	bool	error	error	error	error	error	error	error	error	error	error	error	error
char	int	error	error	error	error	error	error	error	error	bool	bool	error	error
char	float	error	error	error	error	error	error	error	error	bool	bool	error	error
char	char	error	error	error	error	error	error	error	error	bool	bool	error	error
char	string	error	error	error	error	error	error	error	error	error	error	error	error
char	bool	error	error	error	error	error	error	error	error	error	error	error	error
string	int	error	error	error	error	error	error	error	error	error	error	error	error
string	float	error	error	error	error	error	error	error	error	error	error	error	error
string	char	error	error	error	error	error	error	error	error	error	error	error	error
string	string	error	error	error	error	error	error	error	error	bool	bool	error	error
string	bool	error	error	error	error	error	error	error	error	error	error	error	error
bool	int	error	error	error	error	error	error	error	error	error	error	error	error
bool	float	error	error	error	error	error	error	error	error	error	error	error	error
bool	char	error	error	error	error	error	error	error	error	error	error	error	error
bool	string	error	error	error	error	error	error	error	error	error	error	error	error
bool	bool	error	error	error	error	error	error	error	error	error	error	bool	bool

Operando Izquierdo y Operando Derecho: Las filas y columnas de la tabla representan los tipos de datos del operando izquierdo y derecho, respectivamente.

Operaciones: Las celdas de la tabla muestran el resultado de aplicar diferentes operaciones binarias entre los tipos de datos del operando izquierdo y derecho.

Tipos Resultantes:

"error": Indica que la operación no está permitida entre los tipos correspondientes.
"bool": El resultado de la operación es de tipo booleano.
"int": El resultado de la operación es de tipo entero.
"float": El resultado de la operación es de tipo punto flotante.

Reglas Específicas:

Operaciones Aritméticas: Las operaciones de suma, resta, multiplicación, división y módulo están definidas para ciertas combinaciones de tipos numéricos. Por ejemplo, la suma de un entero y un float es un float.

Operaciones de Comparación: Las operaciones de comparación (menor, mayor, menor o igual, mayor o igual, igual, no igual) siempre resultan en un tipo booleano.

Operaciones con Char y String: Las operaciones con tipos de datos "char" y "string" generan errores, ya que no están definidas en este contexto.

Administración de Memoria

La distribución de memoria aquí se realiza en bloques de 2000 unidades para cada tipo de variable y constante. Este esquema facilita la identificación y el aislamiento de diferentes tipos de datos (globales, locales, constantes, temporales) para evitar colisiones y mejorar la eficiencia en la asignación y el acceso a la memoria. Al comenzar en 50000 y finalizar cerca de 90000, se proporciona suficiente espacio para cada tipo, permitiendo una expansión o modificación futura sin reestructurar completamente el esquema de memoria, lo cual es crucial para la escalabilidad y mantenimiento del proyecto.

Tipo de Memoria	Dirección de Inicio
Global_Entero	50000
Global_Flotante	52000
Global_Caracter	54000
Limite_Global	56000
Local_Entero	58000
Local_Flotante	60000
Local_Caracter	62000
Limite_Local	64000
Constante_Bool	66000
Constante_Entero	68000
Constante_Flotante	70000
Constante_Caracter	72000
Constante_Cadena	74000
Limite_Constante	76000
TemporalGlobal_Bool	78000
TemporalGlobal_Entero	80000
TemporalGlobal_Flotante	82000
TemporalGlobal_Caracter	84000
TemporalGlobal_Cadena	86000
Limite_TemporalGlobal	88000
TemporalLocal_Bool	90000
TemporalLocal_Entero	92000
TemporalLocal_Flotante	94000
TemporalLocal_Caracter	96000
TemporalLocal_Cadena	98000
Limite_TemporalLocal	100000

Detalles de la Distribución:

- Variables Globales:

Rango de 50000 a 56000, con bloques separados para enteros, flotantes y caracteres.

- Variables Locales:

Rango de 58000 a 64000, similar en estructura a las variables globales pero destinadas a variables dentro de un ámbito local (como dentro de una función).

- Constantes:

Rango de 66000 a 76000, cubriendo booleanos, enteros, flotantes, caracteres y cadenas. Estas direcciones están reservadas para valores constantes que no cambian durante la ejecución del programa.

- Temporales Globales y Locales:

Rangos de 78000 a 88000 para temporales globales y de 90000 a 100000 para temporales locales. Estos espacios son probablemente usados para almacenar resultados intermedios y temporales durante la ejecución de operaciones complejas.

1.1.4. Descripción de la Máquina Virtual.

- Arquitectura y Administración de memoria.

La máquina virtual en CompilarAD23 es una entidad sofisticada que interpreta y ejecuta cuádruplos generados por el compilador. Su diseño encapsula funcionalidades críticas, incluyendo la ejecución de instrucciones, el manejo de memoria durante la ejecución, y la implementación de operaciones especializadas.

Componentes Clave:

- Módulos Importados:

- sys para interacción con el intérprete de Python.
- matplotlib.pyplot y numpy para visualizaciones y cálculos numéricos.
- random y statistics para operaciones aleatorias y estadísticas.

- Estructura y Pilas:

- Pilas para saltos (pilaSalto) y llamadas a funciones (funcStack).
- Lista de parámetros y un diccionario para funciones (directorioFunciones).
- Referencias a bloques de memoria para variables globales, locales, constantes y temporales.

- Administración de Memoria:

- La memoria se divide en distintos segmentos para tipos de datos y ámbitos (globales, locales, constantes, temporales). La máquina virtual gestiona estos segmentos y asegura el aislamiento y eficiencia en la asignación y acceso de memoria. Esta división facilita el manejo de diferentes tipos de datos y su escalabilidad.

Funcionalidades Principales:

- Ejecución de Cuádruplos:

- Interpretación y ejecución de cuádruplos.
- Manejo de saltos condicionales e incondicionales.
- Realización de operaciones aritméticas, lógicas y de entrada/salida.

- Funciones Especiales:

- Implementación de funciones para conversiones de tipo, operaciones matemáticas y estadísticas, y visualización de datos.

- Operaciones de Memoria:

- Obtención y almacenamiento de valores en diferentes segmentos de memoria.
- Resolución de punteros y gestión de parámetros para funciones.

- Proceso de Ejecución:

- La ejecución se lleva a cabo interpretando cada cuádruplo, donde la máquina virtual realiza las operaciones indicadas, tales como asignaciones, saltos, operaciones aritméticas y lógicas, y llamadas a funciones. El sistema de memoria garantiza que cada operación acceda a la ubicación correcta y maneje los datos adecuadamente.

- Consideraciones de Diseño:

- La arquitectura de la máquina virtual está orientada a maximizar la eficiencia y la claridad en la ejecución del código compilado.

- La gestión de memoria está diseñada para ser flexible y escalable, permitiendo futuras expansiones o modificaciones del lenguaje y su compilador.

Función	Descripción
run	Ejecuta cuádruplos desde una lista de cuádruplos, manejando la lógica de ejecución de cada operación.
obtenerValor	Devuelve el valor de una dirección de memoria, resolviendo punteros y accediendo a la memoria correspondiente.
obtenerParametro	Recupera el valor de un parámetro de una función de la memoria correspondiente.
guardarValor	Guarda un valor en una dirección de memoria especificada, manejando distintos segmentos de memoria.
getPointer	Obtiene la dirección real de un valor, en caso de ser un puntero.
exitFunc	Limpia la memoria utilizada por una función y ajusta los desplazamientos de memoria.
do	Ejecuta operaciones aritméticas y lógicas básicas entre dos operandos.
doSpec	Ejecuta funciones especiales precargadas, realizando operaciones específicas y manejo de visualizaciones.
loadArr	Carga un arreglo de memoria para su uso en funciones especiales.

```
    # Ejecutar cuádruplos desde una lista de cuádruplos
    def run(self, CuadruploList, show):
        self.curr = 0
        while self.curr < len(CuadruploList) and CuadruploList[self.curr].operador !=
"Aceptado":
            Cuadruplo = CuadruploList[self.curr]

            if show:  # Mostrar cuádruplo si es necesario
                print(Cuadruplo, end=" ")
            if Cuadruplo.operando2 == None:
                # Ejecutar operaciones más complejas

                if Cuadruplo.operador == "=":  # Asignación
                    op1 = self.obtenerValor(Cuadruplo.operando1)
                    self.guardarValor(Cuadruplo.resultado, op1)

                elif Cuadruplo.operador == "GOTO":  # Salto incondicional
                    self.curr = Cuadruplo.resultado - 1

                elif Cuadruplo.operador == "GOTOF":  # Salto condicional
                    op1 = self.obtenerValor(Cuadruplo.operando1)
                    if not op1:
                        self.curr = Cuadruplo.resultado - 1
```

```
    # Devuelve el valor de la dirección de memoria apropiada
    def obtenerValor(self, dir):
        dir = self.getPointer(dir)  # Obtiene la dirección real en caso de ser un
puntero

        # Determina de qué bloque de memoria recuperar el valor
        if dir < Limite_Global:
            value = self.MemoriaG.obtenerValor(dir)  # Memoria global
        elif dir < Limite_Local:
            value = self.MemoriaL.obtenerValor(dir)  # Memoria local
        elif dir < Limite_Constante:
            value = self.MemoriaC.obtenerValor(dir)  # Memoria de constantes
            # Convertir cadenas "true" y "false" a valores booleanos
            if value == "true":
                value = True
            elif value == "false":
                value = False
        elif dir < Limite_TemporalLocal:
            value = self.MemoriaT.obtenerValor(dir)  # Memoria resultado
            # Igual que arriba, convertir cadenas a booleanos
            if value == "true":
                value = True
            elif value == "false":
                value = False
        else:
            print(f"{dir} es una dirección de memoria inválida")
            sys.exit()
        return value
```

```
    # Devuelve el valor del parámetro de la dirección apropiada
    def obtenerParametro(self, dir):
        dir = self.getPointer(dir)  # Resuelve si es un puntero

        # Determina de qué bloque de memoria recuperar el valor del parámetro
        if dir < Limite_Global:
            value = self.MemoriaG.obtenerValor(dir)
        elif dir < Limite_Local:
            reqVars = self.directorioFunciones[self.funcStack[-1]]["reqVars"]
            value = self.MemoriaL.obtenerParametro(dir, reqVars)
        elif dir < Limite_Constante:
            value = self.MemoriaC.obtenerValor(dir)
            # Convertir "true" y "false" a booleanos
            if value == "true":
                value = True
            elif value == "false":
                value = False
```

```
    # Guarda un valor en la dirección apropiada
    def guardarValor(self, dir, value):
        dir = self.getPointer(dir)  # Resuelve si es un puntero
        # Guarda el valor en la memoria correspondiente
        if dir < Limite_Global:
            self.MemoriaG.guardarValor(dir, value)
        elif dir < Limite_Local:
            self.MemoriaL.guardarValor(dir, value)
        elif dir < Limite_Constante:
            self.MemoriaC.guardarValor(dir, value)
        elif dir < Limite_TemporalLocal:
            self.MemoriaT.guardarValor(dir, value)
        else:
            print(f"{dir} es una dirección de memoria inválida")
            sys.exit()
```

```
    # Devuelve la dirección real en caso de que sea un puntero
    def getPointer(self, dir):
        if type(dir) == str:  # Si es un puntero, obtiene la dirección real
            dir = int(dir[1:])
            dir = self.obtenerValor(dir)

        dir = int(dir)
        return dir
```

```
    # Devuelve la dirección real en caso de que sea un puntero
    def getPointer(self, dir):
        if type(dir) == str:  # Si es un puntero, obtiene la dirección real
            dir = int(dir[1:])
            dir = self.obtenerValor(dir)

        dir = int(dir)
        return dir

    # Limpia la memoria que ya no se necesita y ajusta los desplazamientos
    def exitFunc(self):
        func = self.funcStack.pop()

        reqresultados = self.directorioFunciones[func]["reqresultados"]
        reqVars = self.directorioFunciones[func]["reqVars"]

        self.MemoriaL.pop(reqVars)
        self.MemoriaT.pop(reqresultados)
        self.curr = self.pilaSalto.pop()
        if len(self.funcStack) > 0:
            reqresultados = self.directorioFunciones[self.funcStack[-
1]]["reqresultados"]
```

```
    # Ejecutar funciones especiales
    def doSpec(self, dir):
        func = self.funcStack.pop()  # Obtener la función especial a ejecutar

        # Convertir a entero y guardar el resultado
        if func == "int":
            x = int(self.obtenerValor(self.params.pop()))  # Convertir el valor a
entero
            self.guardarValor(dir, x)

        # Convertir a flotante y guardar el resultado
        elif func == "float":
            x = float(self.obtenerValor(self.params.pop()))  # Convertir el valor a
flotante
            self.guardarValor(dir, x)

        # Calcular la potencia y guardar el resultado
        elif func == "pow":
            x = float(self.obtenerValor(self.params.pop()))  # Base
            y = float(self.obtenerValor(self.params.pop()))  # Exponente
            self.guardarValor(dir, pow(x, y))

        # Generar un número aleatorio y guardarlo
        elif func == "rand":
            self.guardarValor(dir, random())  # Guarda un número aleatorio

        # Para otras funciones que requieren cargar un arreglo
        else:
            arr = self.loadArr()  # Carga el arreglo necesario para la función

            # Calcular la media y guardar el resultado
            if func == "med":
                self.guardarValor(dir, mean(arr))

            # Encontrar la moda y guardar el resultado
            elif func == "moda":
                self.guardarValor(dir, mode(arr))

            # Calcular la varianza y guardar el resultado
            elif func == "var":
                self.guardarValor(dir, variance(arr))

            # Realizar una regresión lineal y mostrar el Gráfico de Líneas
            elif func == "reg":
                x = [i for i in range(len(arr))]  # Crear una lista de índices
```

```
    def loadArr(self):
        dir = self.params.pop()  # Obtener la dirección del arreglo
        array = []

        # Determinar el contexto actual para cargar el arreglo
        if len(self.funcStack) == 0:
            curr = self.programID
        else:
            curr = self.funcStack[-1]

        keys = list(self.directorioFunciones[curr]["vars"])  # Obtener las claves
de las variables

        # Encontrar el tamaño del arreglo y cargar sus valores
        for key in keys:
            if self.directorioFunciones[curr]["vars"][key].get("dir") == dir:
                arrSize =
self.directorioFunciones[curr]["vars"][key].get("arrSize")
                break

        for i in range(arrSize):
            array.append(self.obtenerValor(dir + i))  # Cargar cada elemento del
arreglo

        return array  # Devolver el arreglo cargado
```

1.1.5. Pruebas del funcionamiento del Lenguaje.

- Casos de Prueba.

```
program prueba1;
vars
int i, j;
char h;
float ancho, largo, area;

main()
{
    %comentario
    ancho = 5.0;
    largo = 3.14;
    area = ancho/largo;
    for i = 1 to 2 do {
        for j = 1 to 3 do{
            write(i," ","Hola",
"Mundo");
        }
        write("Aceptado");
    }
}
```

```
program area;
vars
int ancho, otro;
int alto[5], largo[6];
int i, algo;
char uno, dos;

main()
{
    i = 0;
    while (i < 5) do {
        alto[i] = i;
        largo[i+1] = alto[i] + 1;
        write(largo[i]);
        i = i + 1;
    }

    i = 0;
    while (i < 5) do {
        write(i);
        i = i + 1;
    }

    read(uno, dos);
    write(uno, dos);
```

```
program Fibonacci;
vars
int temp, next, actual, n, i;
main(){
    read(n);
    if (n >= 1) then{
        next = 1;
    }
    while (i <= n) do {
        temp = actual;
        actual = next;
        next = temp + actual;
        i = i + 1;
    }
    write("La posicion ", n, " es igual a ",
temp);
}
```

```
program fibonacciRecursion;
vars
int i, input, output;

function int fib(int n)
vars
int uno, dos;
{
    if (n <= 1) then {
        return(n);
    }
    else{
        uno = fib(n - 1);
        dos = fib(n - 2);
        return(uno + dos);
    }
}

main(){
    read(input);
    output = fib(input);
    write("pos ", input, " es ", output);
}
```

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py .\prueba1.txt
prueba1:
type: void
vars: {'i': {'type': 'int', 'arrSize': 0, 'dir': 50000}, 'j': {'type': 'int', 'arrSize': 0, 'dir': 50001}, 'h': {'type': 'char', 'arrSize': 0, 'dir': 54000}, 'ancho': {'type': 'float', 'arrSize': 0, 'dir': 52000}, 'largo': {'type': 'float', 'arrSize': 0, 'dir': 52001}, 'area': {'type': 'float', 'arrSize': 0, 'dir': 52002}}
```

Archivo: prueba1.txt
'i': Es una variable de tipo entero (int) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 50000.

'j': Es otra variable de tipo entero (int) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 50001.

'h': Es una variable de tipo caracter (char) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 54000.

'ancho': Es una variable de tipo punto flotante (float) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 52000.

'largo': Es otra variable de tipo punto flotante (float) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 52001.

'area': Es una variable de tipo punto flotante (float) sin tamaño de arreglo (arrSize: 0) y su dirección de memoria es 52002.

```
0 [GOTO, None, None, 1]
1 [=, 70000, None, 52000]
2 [=, 70001, None, 52001]
3 [/, 52000, 52001, 94000]
4 [=, 94000, None, 52002]
5 [=, 68000, None, 50000]
6 [<=, 50000, 68001, 90000]
7 [GOTOF, 90000, None, 22]
8 [=, 68000, None, 50001]
9 [<=, 50001, 68002, 90001]
10 [GOTOF, 90001, None, 18]
11 [PRINT, None, None, 50000]
12 [PRINT, None, None, 74000]
13 [PRINT, None, None, 74001]
14 [PRINT, None, None, 74002]
15 [PRINT, None, None, 74003]
16 [+, 50001, 68000, 50001]
17 [GOTO, None, None, 9]
18 [PRINT, None, None, 74004]
19 [PRINT, None, None, 74003]
20 [+, 50000, 68000, 50000]
21 [GOTO, None, None, 6]
22 [DONE, None, None,  ]
```

```
[GOTO, None, None, 1]
[=, 70000, None, 52000]
[=, 70001, None, 52001]
[/, 52000, 52001, 94000]
[=, 94000, None, 52002]
[=, 68000, None, 50000]
[<=, 50000, 68001, 90000]
[GOTOF, 90000, None, 22]
[=, 68000, None, 50001]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 1
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 1
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 1
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 74004] Aceptado
[PRINT, None, None, 74003]
```

```
[+, 50000, 68000, 50000]
[GOTO, None, None, 6]
[<=, 50000, 68001, 90000]
[GOTOF, 90000, None, 22]
[=, 68000, None, 50001]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 2
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 2
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 50000] 2
[PRINT, None, None, 74000]
[PRINT, None, None, 74001] Hola
[PRINT, None, None, 74002] Mundo
[PRINT, None, None, 74003]

[+, 50001, 68000, 50001]
[GOTO, None, None, 9]
[<=, 50001, 68002, 90001]
[GOTOF, 90001, None, 18]
[PRINT, None, None, 74004]
Aceptado
[PRINT, None, None, 74003]

[+, 50000, 68000, 50000]
[GOTO, None, None, 6]
[<=, 50000, 68001, 90000]
[GOTOF, 90000, None, 22]
[DONE, None, None,  ]
```

- **Cuádruplo 1:** [GOTO, None, None, 1]

Descripción: Salto incondicional al cuádruplo 1, iniciando la ejecución del programa.
Significado: Establece el inicio del programa.

- **Cuádruplo 5**: [=, 68000, None, 50000]

Descripción: Asigna el valor de la variable i a 1.
Significado: Inicializa la variable de control del bucle exterior.

- **Cuádruplo 7**: [GOTOF, 90000, None, 22]

Descripción: Salto condicional al cuádruplo 22 si la condición i <= 2 es falsa.
Significado: Verifica si la condición del bucle exterior (for i = 1 to 2) es verdadera.

- **Cuádruplo 11**: [PRINT, None, None, 50000]

Descripción: Imprime el valor de la variable i.
Significado: Muestra el valor de i en la consola.

- **Cuádruplo 14-15**: [PRINT, None, None, 74000], [PRINT, None, None, 74001], [PRINT, None, None, 74002], [PRINT, None, None, 74003]

Descripción: Imprime cadenas de texto.
Significado: Muestra "Hola Mundo" en la consola.

- **Cuádruplo 18**: [PRINT, None, None, 74004]

Descripción: Imprime la cadena "Aceptado".
Significado: Muestra "Aceptado" en la consola.

- **Cuádruplo 20**: [+, 50000, 68000, 50000]

Descripción: Incrementa el valor de i en 1.
Significado: Avanza al siguiente valor de i en el bucle exterior.

- **Cuádruplo 21**: [GOTO, None, None, 6]

Descripción: Salto incondicional al cuádruplo 6.
Significado: Regresa al inicio del bucle exterior.

- **Cuádruplo 22**: [DONE, None, None,]

Descripción: Fin del programa.
Significado: Indica el final del programa.

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py Fibonacci.txt
func:
type: void
vars: {'temp': {'type': 'int', 'arrSize': 0, 'dir': 50000}, 'next': {'type': 'int', 'arrSize': 0, 'dir': 50001}, 'actual': {'type': 'int', 'arrSize': 0, 'dir': 50002}, '
n': {'type': 'int', 'arrSize': 0, 'dir': 50003}, 'i': {'type': 'int', 'arrSize': 0, 'dir': 50004}}
```

Tipo de Programa: void
Variables:

- temp: Entero (int), sin dimensiones, dirección de memoria 50000.
- next: Entero (int), sin dimensiones, dirección de memoria 50001.
- actual: Entero (int), sin dimensiones, dirección de memoria 50002.
- n: Entero (int), sin dimensiones, dirección de memoria 50003.
- i: Entero (int), sin dimensiones, dirección de memoria 50004.

```
0 [GOTO, None, None, 1]
1 [READ, None, None, 50003]
2 [>=, 50003, 68000, 90000]
3 [GOTOF, 90000, None, 5]
4 [=, 68000, None, 50001]
5 [<=, 50004, 50003, 90001]
6 [GOTOF, 90001, None, 14]
7 [=, 50002, None, 50000]
8 [=, 50001, None, 50002]
9 [+, 50000, 50002, 92000]
10 [=, 92000, None, 50001]
11 [+, 50004, 68000, 92001]
12 [=, 92001, None, 50004]
13 [GOTO, None, None, 5]
14 [PRINT, None, None, 74000]
15 [PRINT, None, None, 50003]
16 [PRINT, None, None, 74001]
17 [PRINT, None, None, 50000]
18 [PRINT, None, None, 74002]
19 [DONE, None, None,  ]
```

```
[GOTO, None, None, 1]
[READ, None, None, 50003] 2

[>=, 50003, 68000, 90000]
[GOTOF, 90000, None, 5]
[=, 68000, None, 50001]
[<=, 50004, 50003, 90001]
[GOTOF, 90001, None, 14]
[=, 50002, None, 50000]
[=, 50001, None, 50002]
[+, 50000, 50002, 92000]
[=, 92000, None, 50001]
[+, 50004, 68000, 92001]
[=, 92001, None, 50004]
[GOTO, None, None, 5]
[<=, 50004, 50003, 90001]
[GOTOF, 90001, None, 14]
[=, 50002, None, 50000]
[=, 50001, None, 50002]
[+, 50000, 50002, 92000]
[=, 92000, None, 50001]
[+, 50004, 68000, 92001]
[=, 92001, None, 50004]
[GOTO, None, None, 5]
[<=, 50004, 50003, 90001]
[GOTOF, 90001, None, 14]
[=, 50002, None, 50000]
[=, 50001, None, 50002]
[+, 50000, 50002, 92000]
[=, 92000, None, 50001]
[+, 50004, 68000, 92001]
[=, 92001, None, 50004]
[GOTO, None, None, 5]
[<=, 50004, 50003, 90001]
[GOTOF, 90001, None, 14]
[PRINT, None, None, 74000] La posicion
[PRINT, None, None, 50003] 2
[PRINT, None, None, 74001]  es igual a
[PRINT, None, None, 50000] 1
[PRINT, None, None, 74002]

[DONE, None, None,  ]
```

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py .\area.txt
area:
type: void
vars: {'ancho': {'type': 'int', 'arrSize': 0, 'dir': 50000}, 'otro': {'type': 'int', 'arrSize': 0, 'dir': 50001}, 'alto': {'type': 'int', 'arrSize': 5, 'dir': 50002}, 'l
argo': {'type': 'int', 'arrSize': 6, 'dir': 50007}, 'i': {'type': 'int', 'arrSize': 0, 'dir': 50013}, 'algo': {'type': 'int', 'arrSize': 0, 'dir': 50014}, 'uno': {'type'
: 'char', 'arrSize': 0, 'dir': 54000}, 'dos': {'type': 'char', 'arrSize': 0, 'dir': 54001}}
```

0 [GOTO, None, None, 1]
1 [=, 68000, None, 50013]
2 [<=, 50013, 68001, 90000]
3 [GOTOF, 90000, None, 20]
4 [VER, 50013, None, 5]
5 [+, 68002, 50013, 92000]
6 [=, 50013, None, *92000]
7 [+, 50013, 68003, 92001]
8 [VER, 92001, None, 6]
9 [+, 68004, 92001, 92002]
10 [VER, 50013, None, 5]
11 [+, 68002, 50013, 92003]
12 [+, *92003, 68003, 92004]
13 [=, 92004, None, *92002]
14 [VER, 50013, None, 6]
15 [+, 68004, 50013, 92005]
16 [PRINT, None, None, *92005]
17 [PRINT, None, None, 74000]
18 [+, 50013, 68003, 50013]
19 [GOTO, None, None, 2]
20 [=, 68000, None, 50013]
21 [<, 50013, 68005, 90001]
22 [GOTOF, 90001, None, 28]
23 [PRINT, None, None, 50013]
24 [PRINT, None, None, 74000]
25 [+, 50013, 68003, 92006]
26 [=, 92006, None, 50013]
27 [GOTO, None, None, 21]
28 [READ, None, None, 54000]
29 [READ, None, None, 54001]
30 [PRINT, None, None, 54000]
31 [PRINT, None, None, 54001]
32 [PRINT, None, None, 74000]
33 [/, 50000, 50007, 92007]
34 [+, 50000, 50007, 92008]
35 [*, 68000, 92008, 92009]
36 [>, 92007, 92009, 90002]
37 [&, 90002, 66000, 90003]
38 [PRINT, None, None, 74001]
39 [PRINT, None, None, 90003]
40 [PRINT, None, None, 74000]
41 [DONE, None, None,]

[GOTO, None, None, 1]
[=, 68000, None, 50013]
[<=, 50013, 68001, 90000]
[GOTOF, 90000, None, 20]
[VER, 50013, None, 5]
[+, 68002, 50013, 92000]
[=, 50013, None, *92000]
[+, 50013, 68003, 92001]
[VER, 92001, None, 6]
[+, 68004, 92001, 92002]
[VER, 50013, None, 5]
[+, 68002, 50013, 92003]
[+, *92003, 68003, 92004]
[=, 92004, None, *92002]
[VER, 50013, None, 6]
[+, 68004, 50013, 92005]
[PRINT, None, None, *92005] 0
[PRINT, None, None, 74000]

[+, 50013, 68003, 50013]
[GOTO, None, None, 2]
[<=, 50013, 68001, 90000]
[GOTOF, 90000, None, 20]
[VER, 50013, None, 5]
[+, 68002, 50013, 92000]
[=, 50013, None, *92000]
[+, 50013, 68003, 92001]
[VER, 92001, None, 6]
[+, 68004, 92001, 92002]
[VER, 50013, None, 5]
[+, 68002, 50013, 92003]
[+, *92003, 68003, 92004]
[=, 92004, None, *92002]
[VER, 50013, None, 6]
[+, 68004, 50013, 92005]
[PRINT, None, None, *92005] 3
[PRINT, None, None, 74000]

```
[+, 50013, 68003, 50013]
[GOTO, None, None, 2]
[<=, 50013, 68001, 90000]
[GOTOF, 90000, None, 20]
[VER, 50013, None, 5]
[+, 68002, 50013, 92000]
[=, 50013, None, *92000]
[+, 50013, 68003, 92001]
[VER, 92001, None, 6]
[+, 68004, 92001, 92002]
[VER, 50013, None, 5]
[+, 68002, 50013, 92003]
[+, *92003, 68003, 92004]
[=, 92004, None, *92002]
[VER, 50013, None, 6]
[+, 68004, 50013, 92005]
[PRINT, None, None, *92005] 4
[PRINT, None, None, 74000]

[+, 50013, 68003, 50013]
[GOTO, None, None, 2]
[<=, 50013, 68001, 90000]
[GOTOF, 90000, None, 20]
[=, 68000, None, 50013]
[<, 50013, 68005, 90001]
[GOTOF, 90001, None, 28]
[PRINT, None, None, 50013] 2
[PRINT, None, None, 74000]

[+, 50013, 68003, 92006]
[=, 92006, None, 50013]
[GOTO, None, None, 21]
[<, 50013, 68005, 90001]
[GOTOF, 90001, None, 28]
[PRINT, None, None, 50013] 3
[PRINT, None, None, 74000]

[+, 50013, 68003, 92006]
[=, 92006, None, 50013]
[GOTO, None, None, 21]
[<, 50013, 68005, 90001]
[GOTOF, 90001, None, 28]
[PRINT, None, None, 50013] 4
[PRINT, None, None, 74000]

[+, 50013, 68003, 92006]
[=, 92006, None, 50013]
[GOTO, None, None, 21]
[<, 50013, 68005, 90001]
[GOTOF, 90001, None, 28]
[READ, None, None, 54000] 4
```

```
[READ, None, None, 54001] 1

[PRINT, None, None, 54000] 4
[PRINT, None, None, 54001] 1
[PRINT, None, None, 74000]

[/, 50000, 50007, 92007] Error: no se
puede dividir por 0
```

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py .\fibonacciRecursion.txt
fibonacciRecursion:
type: void
vars: {'i': {'type': 'int', 'arrSize': 0, 'dir': 50000}, 'input': {'type': 'int', 'arrSize': 0, 'dir': 50001}, 'output': {'type': 'int', 'arrSize': 0, 'dir': 50002}, 'fi
b': {'type': 'int', 'arrSize': 0, 'dir': 50003}}

fib:
type: int
dir: 1
vars: {'n': {'type': 'int', 'arrSize': 0, 'dir': 58000}, 'uno': {'type': 'int', 'arrSize': 0, 'dir': 58001}, 'dos': {'type': 'int', 'arrSize': 0, 'dir': 58002}}
params: 1
reqresultados: {'bool': 1, 'int': 5, 'float': 0, 'char': 0, 'string': 0}
reqVars: {'int': 3, 'float': 0, 'char': 0}
```

fibonacciRecursion.txt:

0 [GOTO, None, None, 20]
1 [<=, 58000, 68000, 78000]
2 [GOTOF, 78000, None, 5]
3 [RETURN, None, None, 58000]
4 [GOTO, None, None, 19]
5 [-, 58000, 68000, 80000]
6 [ERA, None, None, fib]
7 [PARAM, 80000, None, 0]
8 [GOSUB, None, None, 1]
9 [=, 50003, None, 80001]
10 [=, 80001, None, 58001]
11 [-, 58000, 68001, 80002]
12 [ERA, None, None, fib]
13 [PARAM, 80002, None, 0]
14 [GOSUB, None, None, 1]
15 [=, 50003, None, 80003]
16 [=, 80003, None, 58002]
17 [+, 58001, 58002, 80004]
18 [RETURN, None, None, 80004]
19 [ENDFUNC, None, None, fib]
20 [READ, None, None, 50001]
21 [ERA, None, None, fib]
22 [PARAM, 50001, None, 0]
23 [GOSUB, None, None, 1]
24 [=, 50003, None, 92000]
25 [=, 92000, None, 50002]
26 [PRINT, None, None, 74000]
27 [PRINT, None, None, 50001]
28 [PRINT, None, None, 74001]
29 [PRINT, None, None, 50002]
30 [PRINT, None, None, 74002]
31 [DONE, None, None,]

Tipo de Programa: void
Variables:
i: Entero (int), sin dimensiones, dirección de memoria 50000.
input: Entero (int), sin dimensiones, dirección de memoria 50001.
output: Entero (int), sin dimensiones, dirección de memoria 50002.
fib: Función de tipo entero (int), sin dimensiones, dirección de memoria 50003.
fib:

Tipo de Función: int
Dirección de Memoria: 1
Variables:
n: Entero (int), sin dimensiones, dirección de memoria 58000.
uno: Entero (int), sin dimensiones, dirección de memoria 58001.
dos: Entero (int), sin dimensiones, dirección de memoria 58002.
Parámetros: 1
Requiere Resultados:
bool: 1
int: 5
float: 0
char: 0
string: 0
Requiere Variables:
int: 3
float: 0
char: 0

```
[GOTO, None, None, 20]
[READ, None, None, 50001] 3

[ERA, None, None, fib]
[PARAM, 50001, None, 0]
[GOSUB, None, None, 1]
[<=, 58000, 68000, 78000]
[GOTOF, 78000, None, 5]
[-, 58000, 68000, 80000]
[ERA, None, None, fib]
[PARAM, 80000, None, 0]
[GOSUB, None, None, 1]
[<=, 58000, 68000, 78000]
[GOTOF, 78000, None, 5]
[-, 58000, 68000, 80000]
[ERA, None, None, fib]
[PARAM, 80000, None, 0]
[GOSUB, None, None, 1]
[<=, 58000, 68000, 78000]
[GOTOF, 78000, None, 5]
[RETURN, None, None, 58000]
[=, 50003, None, 80001]
[=, 80001, None, 58001]
[-, 58000, 68001, 80002]
[ERA, None, None, fib]
[PARAM, 80002, None, 0]
[GOSUB, None, None, 1]
[<=, 58000, 68000, 78000]
[GOTOF, 78000, None, 5]
[RETURN, None, None, 58000]
[=, 50003, None, 80003]
[=, 80003, None, 58002]
[+, 58001, 58002, 80004]
[RETURN, None, None, 80004]
[=, 50003, None, 80001]
[=, 80001, None, 58001]
[-, 58000, 68001, 80002]
[ERA, None, None, fib]
[PARAM, 80002, None, 0]
[GOSUB, None, None, 1]
[<=, 58000, 68000, 78000]
[GOTOF, 78000, None, 5]
[RETURN, None, None, 58000]
[=, 50003, None, 80003]
[=, 80003, None, 58002]
[+, 58001, 58002, 80004]
[RETURN, None, None, 80004]
[=, 50003, None, 92000]
[=, 92000, None, 50002]
[PRINT, None, None, 74000] pos
[PRINT, None, None, 50001] 3
[PRINT, None, None, 74001]  es
[PRINT, None, None, 50002] 2
[PRINT, None, None, 74002]

[DONE, None, None,  ]
```

```
program FuncPrueba;
vars
int array[8], arrC[8], i;
float f, arrB[8];

function void testOnFunc(int arr[8])
{

    i = int(3.6);
    write(i);

    f = float(4);
    write(f);

    f = pow(1.0, 2);
    write(f);

    f = rand();
    write(f);

    f = med(arr);
    write(f);

    f = moda(arr);
    write(f);

    f = var(arr);
    write(f);

    write(arr);

    reg(arr);

    plot(arr);
}


main(){
    %load arrays
    for i = 0 to 9 do {
        f = rand();
        array[i] = int(f * 8);

        f = rand();
        arrB[i] = f * 8;
    }
    testOnFunc(array);

    i = int(4.6);
    write(i);
    i = int(5.4);
    write(i);

    f = float(5);
    write(f);

    f = pow(1.0, 2);
    write(f);

    f = rand();
    write(f);

    f = med(array);
    write(f);

    f = moda(array);
    write(f);

    f = var(array);
    write(f);

    write(array);
    write(arrB);

    reg(array);
    reg(arrB);

    plot(array);
    plot(arrB);
}
```

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py .\FuncPrueba.txt
Illegal character F in line 1
uncPrueba:
type: void
vars: {'array': {'type': 'int', 'arrSize': 8, 'dir': 50000}, 'arrC': {'type': 'int', 'arrSize': 8, 'dir': 50008}, 'i': {'type': 'int', 'arrSize': 0, 'dir': 50016}, 'f':
{'type': 'float', 'arrSize': 0, 'dir': 52000}, 'arrB': {'type': 'float', 'arrSize': 8, 'dir': 52001}}

testOnFunc:
type: void
dir: 1
vars: {'arr': {'type': 'int', 'arrSize': 8, 'dir': 58000}}
params: 1
reqresultados: {'bool': 0, 'int': 1, 'float': 6, 'char': 0, 'string': 0}
reqVars: {'int': 8, 'float': 0, 'char': 0}
```

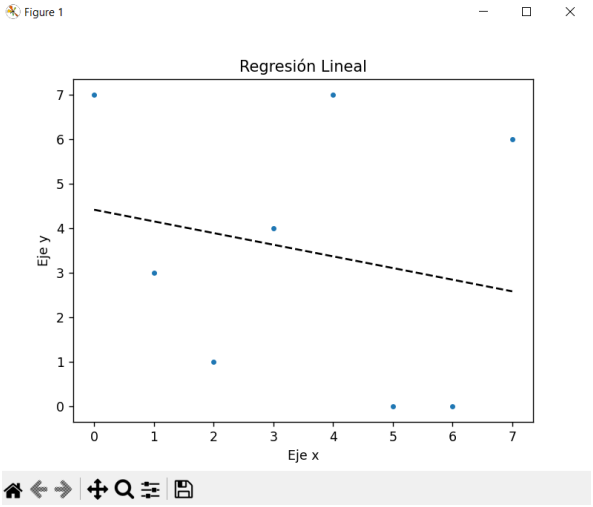
0 [GOTO, None, None, 69]
1 [ERA, None, None, int]
2 [PARAM, 70000, None, 0]
3 [GOSUB, None, None, spec]
4 [=, 80000, None, 50016]
5 [PRINT, None, None, 50016]
6 [PRINT, None, None, 74000]
7 [ERA, None, None, float]
8 [PARAM, 68000, None, 0]
9 [GOSUB, None, None, spec]
10 [=, 82000, None, 52000]
11 [PRINT, None, None, 52000]
12 [PRINT, None, None, 74000]
13 [ERA, None, None, pow]
14 [PARAM, 70001, None, 0]
15 [PARAM, 68001, None, 1]
16 [GOSUB, None, None, spec]
17 [=, 82001, None, 52000]
18 [PRINT, None, None, 52000]
19 [PRINT, None, None, 74000]
20 [ERA, None, None, rand]
21 [GOSUB, None, None, spec]
22 [=, 82002, None, 52000]
23 [PRINT, None, None, 52000]
24 [PRINT, None, None, 74000]
25 [ERA, None, None, med]
26 [PARAM, 58000, None, 0]
27 [GOSUB, None, None, spec]
28 [=, 82003, None, 52000]
29 [PRINT, None, None, 52000]
30 [PRINT, None, None, 74000]
31 [ERA, None, None, moda]
32 [PARAM, 58000, None, 0]
33 [GOSUB, None, None, spec]
34 [=, 82004, None, 52000]
35 [PRINT, None, None, 52000]
36 [PRINT, None, None, 74000]
37 [ERA, None, None, var]
38 [PARAM, 58000, None, 0]
39 [GOSUB, None, None, spec]
40 [=, 82005, None, 52000]

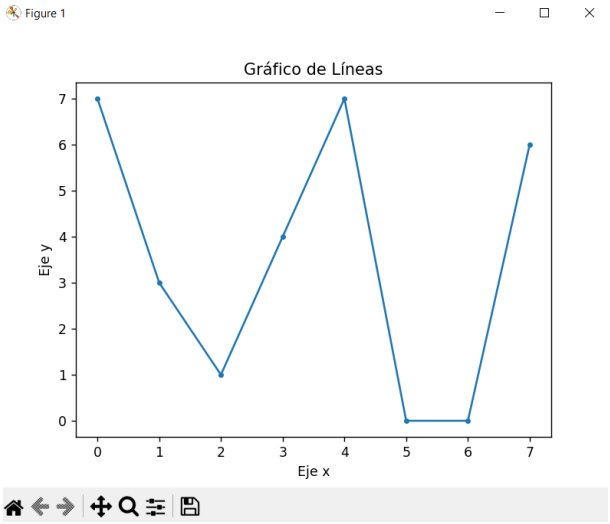
41 [PRINT, None, None, 52000]
42 [PRINT, None, None, 74000]
43 [PRINT, None, None, 74001]
44 [PRINT, None, None, 58000]
45 [PRINT, None, None, 74002]
46 [PRINT, None, None, 58001]
47 [PRINT, None, None, 74002]
48 [PRINT, None, None, 58002]
49 [PRINT, None, None, 74002]
50 [PRINT, None, None, 58003]
51 [PRINT, None, None, 74002]
52 [PRINT, None, None, 58004]
53 [PRINT, None, None, 74002]
54 [PRINT, None, None, 58005]
55 [PRINT, None, None, 74002]
56 [PRINT, None, None, 58006]
57 [PRINT, None, None, 74002]
58 [PRINT, None, None, 58007]
59 [PRINT, None, None, 74002]
60 [PRINT, None, None, 74003]
61 [PRINT, None, None, 74000]
62 [ERA, None, None, reg]
63 [PARAM, 58000, None, 0]
64 [GOSUB, None, None, spec]
65 [ERA, None, None, plot]
66 [PARAM, 58000, None, 0]
67 [GOSUB, None, None, spec]
68 [ENDFUNC, None, None, testOnFunc]
69 [=, 68002, None, 50016]
70 [<=, 50016, 68003, 90000]
71 [GOTOF, 90000, None, 91]
72 [ERA, None, None, rand]
73 [GOSUB, None, None, spec]
74 [=, 94000, None, 52000]
75 [VER, 50016, None, 8]
76 [+, 68004, 50016, 92000]
77 [*, 52000, 68005, 94001]

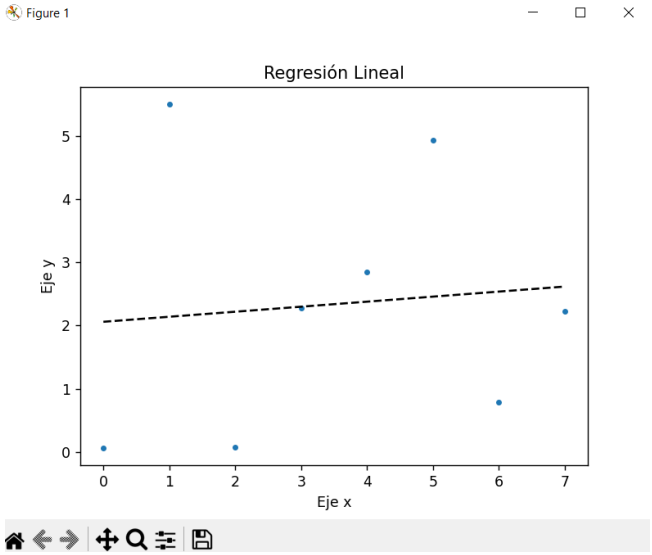
```
78 [ERA, None, None, int]
79 [PARAM, 94001, None, 0]
80 [GOSUB, None, None, spec]
81 [=, 92001, None, *92000]
82 [ERA, None, None, rand]
83 [GOSUB, None, None, spec]
84 [=, 94002, None, 52000]
85 [VER, 50016, None, 8]
86 [+, 70002, 50016, 94003]
87 [*, 52000, 68005, 94004]
88 [=, 94004, None, *94003]
89 [+, 50016, 70001, 50016]
90 [GOTO, None, None, 70]
91 [ERA, None, None, testOnFunc]
92 [PARAM, 50000, None, 0]
93 [GOSUB, None, None, 1]
94 [ERA, None, None, int]
95 [PARAM, 70000, None, 0]
96 [GOSUB, None, None, spec]
97 [=, 92002, None, 50016]
98 [PRINT, None, None, 50016]
99 [PRINT, None, None, 74000]
100 [ERA, None, None, int]
101 [PARAM, 70003, None, 0]
102 [GOSUB, None, None, spec]
103 [=, 92003, None, 50016]
104 [PRINT, None, None, 50016]
105 [PRINT, None, None, 74000]
106 [ERA, None, None, float]
107 [PARAM, 68000, None, 0]
108 [GOSUB, None, None, spec]
109 [=, 94005, None, 52000]
110 [PRINT, None, None, 52000]
111 [PRINT, None, None, 74000]
112 [ERA, None, None, pow]
113 [PARAM, 70001, None, 0]
114 [PARAM, 68001, None, 1]
115 [GOSUB, None, None, spec]
116 [=, 94006, None, 52000]
117 [PRINT, None, None, 52000]
118 [PRINT, None, None, 74000]
119 [ERA, None, None, rand]
120 [GOSUB, None, None, spec]
121 [=, 94007, None, 52000]
122 [PRINT, None, None, 52000]
123 [PRINT, None, None, 74000]
124 [ERA, None, None, med]
125 [PARAM, 50000, None, 0]
126 [GOSUB, None, None, spec]
127 [=, 94008, None, 52000]
```

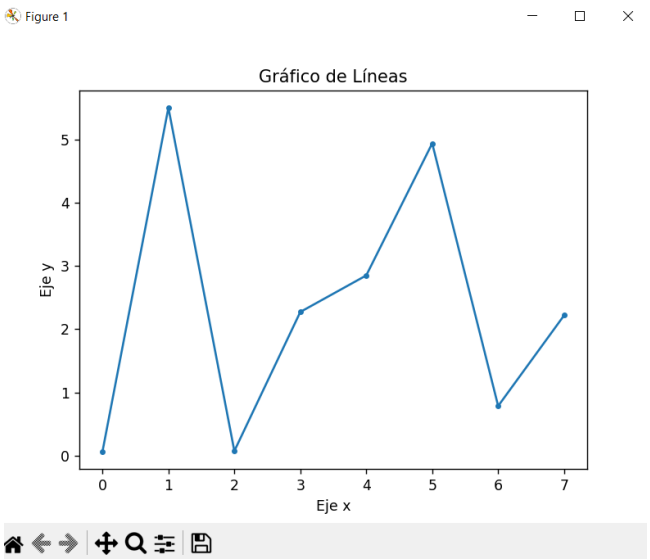
```
128 [PRINT, None, None, 52000]
129 [PRINT, None, None, 74000]
130 [ERA, None, None, moda]
131 [PARAM, 50000, None, 0]
132 [GOSUB, None, None, spec]
133 [=, 94009, None, 52000]
134 [PRINT, None, None, 52000]
135 [PRINT, None, None, 74000]
136 [ERA, None, None, var]
137 [PARAM, 50000, None, 0]
138 [GOSUB, None, None, spec]
139 [=, 94010, None, 52000]
140 [PRINT, None, None, 52000]
141 [PRINT, None, None, 74000]
142 [PRINT, None, None, 74001]
143 [PRINT, None, None, 50000]
144 [PRINT, None, None, 74002]
145 [PRINT, None, None, 50001]
146 [PRINT, None, None, 74002]
147 [PRINT, None, None, 50002]
148 [PRINT, None, None, 74002]
149 [PRINT, None, None, 50003]
150 [PRINT, None, None, 74002]
151 [PRINT, None, None, 50004]
152 [PRINT, None, None, 74002]
153 [PRINT, None, None, 50005]
154 [PRINT, None, None, 74002]
155 [PRINT, None, None, 50006]
156 [PRINT, None, None, 74002]
157 [PRINT, None, None, 50007]
158 [PRINT, None, None, 74002]
159 [PRINT, None, None, 74003]
160 [PRINT, None, None, 74000]
161 [PRINT, None, None, 74001]
162 [PRINT, None, None, 52001]
163 [PRINT, None, None, 74002]
164 [PRINT, None, None, 52002]
165 [PRINT, None, None, 74002]
166 [PRINT, None, None, 52003]
167 [PRINT, None, None, 74002]
168 [PRINT, None, None, 52004]
169 [PRINT, None, None, 74002]
170 [PRINT, None, None, 52005]
171 [PRINT, None, None, 74002]
172 [PRINT, None, None, 52006]
173 [PRINT, None, None, 74002]
174 [PRINT, None, None, 52007]
175 [PRINT, None, None, 74002]
176 [PRINT, None, None, 52008]
177 [PRINT, None, None, 74002]
178 [PRINT, None, None, 74003]
179 [PRINT, None, None, 74000]
```

```
180 [ERA, None, None, reg]
181 [PARAM, 50000, None, 0]
182 [GOSUB, None, None, spec]
183 [ERA, None, None, reg]
184 [PARAM, 52001, None, 0]
185 [GOSUB, None, None, spec]
186 [ERA, None, None, plot]
187 [PARAM, 50000, None, 0]
188 [GOSUB, None, None, spec]
189 [ERA, None, None, plot]
190 [PARAM, 52001, None, 0]
191 [GOSUB, None, None, spec]
192 [DONE, None, None,  ]
```










```
[PRINT, None, None, 74000]

[ERA, None, None, reg]
[PARAM, 50000, None, 0]
[GOSUB, None, None, spec]
[ERA, None, None, reg]
[PARAM, 52001, None, 0]
[GOSUB, None, None, spec]
[ERA, None, None, plot]
[PARAM, 50000, None, 0]
[GOSUB, None, None, spec]
[ERA, None, None, plot]
[PARAM, 52001, None, 0]
[GOSUB, None, None, spec]
[DONE, None, None,  ]
```

1.1.6. Documentación del Código del Proyecto.

- Explicaciones del Código Fuente.

```
# Por otro lado, la clase MemoriaLocal maneja la asignación y liberación de memoria local, así como la gestión del ámbito.

# La función asignarMemoria asigna memoria para nuevas variables locales, ajustando el desplazamiento según el tipo y el tamaño del arreglo. Las funciones clear, era, pop y reajustarDesplazamiento gestionan la entrada y salida de ámbitos locales, limpiando la memoria no utilizada y ajustando el desplazamiento.

# En ambas clases, la coherencia en el manejo de direcciones de memoria y tipos de datos es esencial para garantizar la correcta ejecución del compilador y la generación de código intermedio.
# Clase diseñada para la gestión de variables locales

class MemoriaLocal:
    def __init__(self) -> None:
        # Inicialización de contadores y listas para la gestión de la memoria local.
        self.contadorEnteros = 0
        self.listaEnteros = []
        self.contadorFlotantes = 0
        self.listaFlotantes = []
        self.contadorCaracteres = 0
        self.listaCaracteres = []
        # Mapa de desplazamiento para manejar el offset de las variables locales.
        self.varOffsetMap = {"int": 0, "float": 0, "char": 0}

    def asignarMemoria(self, var):
        # Asigna memoria para una nueva variable local según su tipo y tamaño.
        varType = var.get("type")
…….
```

Inicialización de la Clase:
La clase tiene atributos para contadores y listas de variables temporales globales y locales, junto con un mapa de desplazamiento temporal para rastrear la asignación de memoria.

Método asignarMemoria:
Asigna memoria para variables temporales, ya sea locales o globales, según el tipo de variable.
Verifica la disponibilidad de espacio y asigna memoria en consecuencia.

Método clear:
Limpia y reinicia la memoria temporal local entre funciones.
Devuelve la memoria requerida actualmente.

Método era:
Asigna espacio e inicializa temporales locales, guardando el desplazamiento.
Configura el mapa de desplazamiento de memoria temporal.

Método pop:
Libera memoria no requerida y restablece los desplazamientos a su estado anterior.

Método reajustarDesplazamiento:
Revierte el desplazamiento de las direcciones de memoria temporales.

Métodos obtenerValor, obtenerParametro, guardarValor:
Realizan operaciones para obtener y almacenar valores en direcciones de memoria especificadas.
Consideran si las direcciones son para temporales locales o globales.

```
# Gestor de memoria temporal
class MemoriaTemporal:
    def __init__(self) -> None:
        # Inicialización de contadores para temporales globales.
        self.contadorGlobalBool = 0
        self.contadorGlobalEntero = 0
        self.contadorGlobalFlotante = 0
        self.contadorGlobalCaracter = 0
        self.contadorGlobalCadena = 0

        # Inicialización de listas para almacenar temporales globales.
        self.listaGlobalBool = []
        self.listaGlobalEntero = []
        self.listaGlobalFlotante = []
        self.listaGlobalCaracter = []
        self.listaGlobalCadena = []

        # Inicialización de contadores para temporales locales.
        self.contadorLocalBool = 0
        self.contadorLocalEntero = 0
        self.contadorLocalFlotante = 0
        self.contadorLocalCaracter = 0
        self.contadorLocalCadena = 0
……..
```

Inicialización de la Clase:
La clase tiene contadores e inicialización de listas para cada tipo de constante: booleanas, enteras, flotantes, caracteres y cadenas de texto.

Método asignarMemoria:
Asigna memoria para constantes basándose en su tipo y valor.
Calcula la dirección de memoria dependiendo del tipo y del contador respectivo.
Verifica la disponibilidad de espacio y asigna memoria en consecuencia.
Almacena el valor de la constante en la lista correspondiente.

Método obtenerValor:
Recupera el valor almacenado en una dirección de memoria específica para constantes.
Verifica si la dirección está en el rango válido para constantes.
Recupera el valor según el tipo de constante y ajusta la dirección con el contador correspondiente.

```
# Gestiona la memoria asignada a constantes de diferentes tipos.
class MemoriaConst:
    def __init__(self) -> None:
        # Contadores e inicialización de listas para cada tipo de constante.
        self.boolCount = 0
        self.boolList = []
        self.contadorEnteros = 0
        self.listaEnteros = []
        self.contadorFlotantes = 0
        self.listaFlotantes = []
        self.contadorCaracteres = 0
        self.listaCaracteres = []
        self.stringCount = 0
        self.stringList = []

    def asignarMemoria(self, var):
        # Asigna memoria para constantes, basándose en su tipo y valor.
        varType = var.get("type")
        value = var.get("id")

        # Asignación de memoria para constantes booleanas.
        if varType == "bool":
            dir = Constante_Bool
            dir += self.boolCount
            if dir < Constante_Bool or dir >= Constante_Entero:
                print("no memory available")
                sys.exit()
            else:
                self.boolCount += 1
                self.boolList.append(value)

        # Asignación de memoria para constantes enteras.
        elif varType == "int":
            dir = Constante_Entero
            dir += self.contadorEnteros
            if dir < Constante_Entero or dir >= Constante_Flotante:
……
```

```
# Este código implementa dos clases, MemoriaGlobal y MemoriaLocal, que son esenciales para la gestión de la memoria global y local en un contexto de compilación de lenguaje de programación.

# La clase MemoriaGlobal se encarga de asignar y manipular la memoria global para variables de diferentes tipos, inicializando contadores y listas para cada tipo de variable global.

# La función asignarMemoria se utiliza para asignar memoria a nuevas variables globales, considerando el tipo y el tamaño del arreglo. Además, las funciones obtenerValor, guardarValor y read permiten acceder y modificar los valores almacenados en direcciones de memoria específicas.

# Clase para la gestión de memoria global.
class MemoriaGlobal:
    def __init__(self) -> None:
        # Inicializa contadores y listas para cada tipo de variable global.
        self.contadorEnteros = 0
        self.listaEnteros = []
        self.contadorFlotantes = 0
        self.listaFlotantes = []
        self.contadorCaracteres = 0
        self.listaCaracteres = []

    # Asignación de memoria para nuevas variables globales.
    def asignarMemoria(self, var):
        # Identifica el tipo de variable y asigna memoria adecuadamente.
        varType = var.get("type")
        # Asignación de memoria para enteros.
        if varType == "int":
            dir = Global_Entero
            arrSize = int(var.get("arrSize"))
            dir += self.contadorEnteros + arrSize
            # Verifica si hay espacio disponible en la memoria global para enteros.
            if dir < Global_Entero or dir >= Global_Flotante:
                print("no global memory for int variables available")
                sys.exit()
            else:
                # Asigna memoria y actualiza el contador de enteros.
                if arrSize > 1:
                    self.contadorEnteros += arrSize
                    dir -= arrSize
                    for i in range(arrSize):
                        self.listaEnteros.append(0)
                else:
                    self.contadorEnteros += 1
                    self.listaEnteros.append(0)

        # Asignación de memoria para flotantes.
        elif varType == "float":
            dir = Global_Flotante
            arrSize = int(var.get("arrSize"))
……….
```

1.1.7. Manual de Usuario.

- Guía de Uso.

Manual de Uso del Lenguaje CompilarAD23

Antes de Comenzar

- ¡Requisitos e Instalación!

Python: CompilarAD23 se ejecuta sobre Python, por lo que es esencial tener una versión actualizada de Python instalada.

- Descargar el repositorio a través de las siguientes 2 opciones:

Drive(ZIP): https://drive.google.com/file/d/19FxrrTl-Fp6FFlBwXpmKXa5gdCv-L-zL/view?usp=drive_link

Repositorio_GitHub: https://github.com/YanniFM/CompilarAD23.git

- Librerías Necesarias:

ply: para el análisis léxico y sintáctico.

numpy: para cálculos y manipulaciones numéricas avanzadas.

matplotlib.pyplot: para visualizaciones y gráficos.

(**Nota:** Los siguientes pasos solo son demostrativos hipotéticos en caso de que por alguna razón no se encuentren las librerías ya dentro del proyecto)

Paso 1: Verificar la Instalación de Python y Pip

Antes de instalar las librerías, asegúrate de tener Python y Pip (Python package manager) instalados. Puedes verificar esto ejecutando los siguientes comandos en la terminal de VS Code:

python --version

pip –version

Paso 2: Abrir la Terminal en VS Code

Abre la terminal integrada en VS Code. Puedes hacerlo con el atajo de teclado Ctrl + ñ (en Windows/Linux) o Command + ñ (en macOS), o yendo al menú Ver > Terminal.

Paso 3: Instalar las Librerías

Utiliza los siguientes comandos en la terminal para instalar las librerías:

Para instalar PLY:

pip install ply

Para instalar NumPy:

pip install numpy

Para instalar Matplotlib:

pip install matplotlib

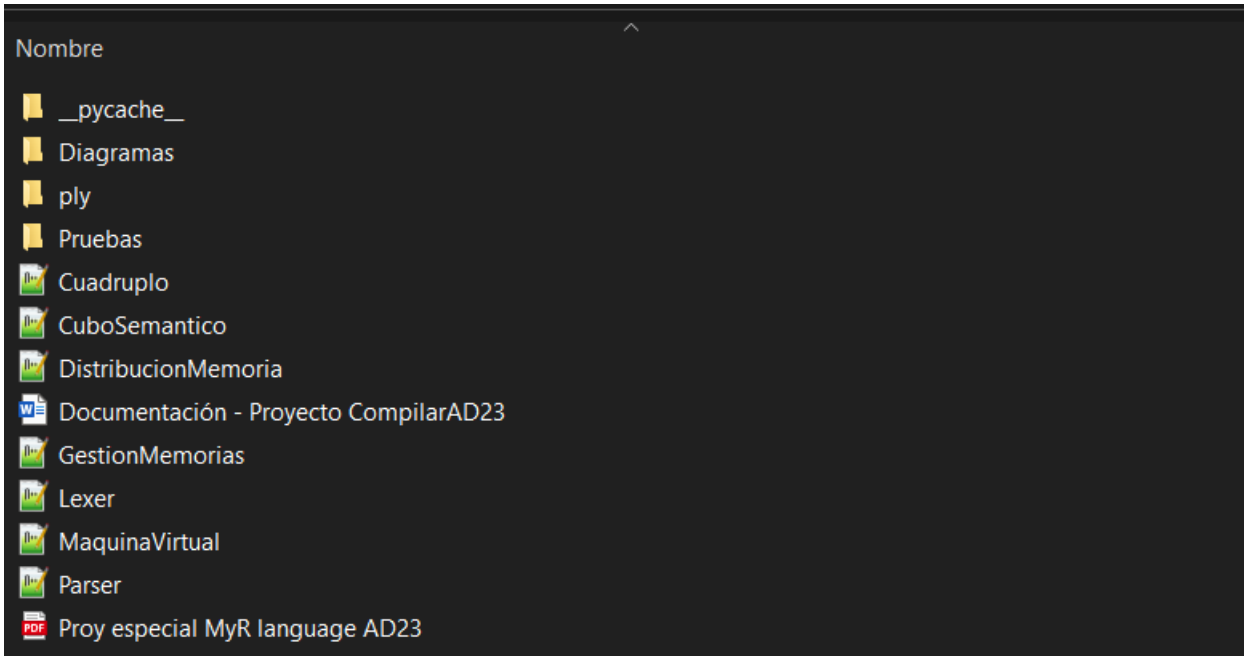
Revisión de la Instalación:

Después de la instalación, puedes verificar que las librerías se han instalado correctamente ejecutando:

pip list

(Esto mostrará una lista de todas las librerías instaladas junto con sus versiones.)

Una vez se tenga posición del proyecto completo se puede continuar a desplazar los archivos de prueba que se quieran ejecutar dentro de la dirección de la carpeta inicial CompilarAD23 que es donde deberían mostrarse los siguientes archivos:



Para empezar a correr el programa, basta con abrir el archivo Parser.py y desde la terminal se correra el siguiente comando:

python Parser.py <Nombredelarchivo.txt>

Ejemplo: python Parser.py test0.txt

```
PS C:\Users\yan-3\OneDrive\Escritorio\ITESM\Semestre 2023-2\Compiladores\Clase\CompilarAD23> python Parser.py test0.txt
area:
type: void
vars: {'ancho': {'type': 'int', 'arrSize': 0, 'dir': 50000}, 'otro': {'type': 'int', 'arrSize': 0, 'dir': 50001}, 'alto': {'type': 'int', 'arrSize': 5, 'dir': 50002}, 'l
argo': {'type': 'int', 'arrSize': 6, 'dir': 50007}, 'i': {'type': 'int', 'arrSize': 0, 'dir': 50013}, 'algo': {'type': 'int', 'arrSize': 0, 'dir': 50014}, 'uno': {'type'
: 'char', 'arrSize': 0, 'dir': 54000}, 'dos': {'type': 'char', 'arrSize': 0, 'dir': 54001}}

0 [GOTO, None, None, 1]
1 [=, 68000, None, 50013]
2 [<=, 50013, 68001, 90000]
3 [GOTOF, 90000, None, 20]
4 [VER, 50013, None, 5]
5 [+, 68002, 50013, 92000]
6 [=, 50013, None, *92000]
7 [+, 50013, 68003, 92001]
8 [VER, 92001, None, 6]
9 [+, 68004, 92001, 92002]
10 [VER, 50013, None, 5]
```

Conclusiones y Reflexiones.

- Reflexiones finales sobre el proyecto y aprendizajes.

Reflexión Final sobre el Proyecto CompilarAD23 y Aprendizajes

Abordar el proyecto CompilarAD23 ha sido una experiencia desafiante y enriquecedora a partes iguales. Al principio, encontré desalentador el proceso de transformar los conceptos teóricos de clase en código funcional. No estaba acostumbrado a emprender proyectos de esta magnitud por mi cuenta, y la naturaleza exhaustiva del proyecto me pareció abrumadora. Sin embargo, esta experiencia me brindó una perspectiva única sobre mis habilidades y áreas de mejora.

El acceso a diversas guías en línea, recursos de clase y, lo más importante, la asistencia de ChatGPT como mentor, fueron elementos cruciales en mi proceso de autoaprendizaje. Estos recursos me proporcionaron la orientación y el apoyo necesarios para navegar por las complejidades del proyecto. A pesar de enfrentar retrasos en algunas entregas debido a mi poca adaptación, considero que al final pude progresar significativamente. Este aspecto, en particular, me enseñó la importancia de la adaptabilidad y la resiliencia frente a proyectos desafiantes.

En el transcurso del proyecto, pude practicar y aplicar diversas prácticas estándar que se discutieron en clase. Esta aplicación práctica no solo reforzó mi comprensión de los conceptos teóricos, sino que también enriqueció mi conjunto de habilidades técnicas.

Finalmente, la materia de Diseño de Compiladores se destacó como una de las más desafiantes de mi carrera. Esta asignatura me permitió comprender algunas mecánicas detrás de los lenguajes de programación, un conocimiento que considero valioso y aplicable en mi futuro profesional. Aunque enfrenté dificultades iniciales y momentos de incertidumbre, decidí no desmoralizarme, aun por más imposible que se me presentaba la situación, creo que fue sabio seguir buscando la manera correcta de seguir avanzando y aprendiendo a una forma que se moldeara conmigo.

1.1.8. Referencias.

- Listado de fuentes y recursos utilizados.

- **PLY (Python Lex-Yacc):**

Documentación Oficial de PLY: PLY Documentation
PLY es una implementación de lex y yacc en Python, utilizada para el análisis léxico y sintáctico en el proyecto. https://www.dabeaz.com/ply/ply.html

- **Python Official Documentation:**

Documentación de Python: Python Docs
La documentación oficial de Python fue esencial para entender las características del lenguaje utilizadas en el desarrollo del compilador. https://docs.python.org/3/

- **NumPy Documentation:**

Guía de NumPy: NumPy Docs
NumPy es utilizado en la máquina virtual para operaciones matemáticas y estadísticas.
https://numpy.org/doc/

- **Matplotlib Documentation:**

Documentación de Matplotlib: Matplotlib Docs
Matplotlib se utiliza en la máquina virtual para la generación de visualizaciones y gráficos. Recursos Educativos y Tutoriales
https://matplotlib.org/stable/users/index.html

- **Estructuras de Datos y Algoritmos en Python:**

Recursos para estructuras de datos: Data Structures in Python
Este recurso ofrece una sólida base en estructuras de datos y algoritmos, cruciales para la implementación de compiladores. https://runestone.academy/ns/books/published/pythonds/index.html

- **Git y GitHub:**

Para control de versiones y colaboración: GitHub
Git y GitHub fueron utilizados para la gestión del código fuente y la colaboración del equipo.
https://github.com/YanniFM/CompilarAD23.git

- **Visual Studio Code:**

Editor de código: Visual Studio Code
Visual Studio Code fue el editor principal para el desarrollo del proyecto, proporcionando un entorno de codificación eficiente. https://code.visualstudio.com/