MANUAL DE USUARIO
Actividad de Aprendizaje - Proyecto Final

Teoría de la Computación

Maestro: Luis Fernando Curi Quintal

Integrantes del equipo:

- Canul Duran Nicolás Israel
- Couoh Concha Valeria Fernanda
- Gonzalez Alcocer Fabio Leonardo
- Gonzalez Chable Said Alfredo

Índice

1. Presentación del Proyecto

El presente documento constituye el **Manual de Usuario del Compilador MIO**, un proyecto académico desarrollado para la materia *Teoría de la Computación*, cuyo objetivo es poner en práctica los principios fundamentales del análisis léxico y sintáctico mediante la implementación de un compilador funcional para un lenguaje de programación diseñado específicamente con fines educativos.

El lenguaje **MIO** es un lenguaje estructurado con sintaxis clara y acotada, creado para ejemplificar el funcionamiento interno de un compilador real: desde el reconocimiento básico de símbolos y patrones (análisis léxico), hasta la verificación de estructuras válidas de acuerdo con una gramática formal (análisis sintáctico).

El compilador desarrollado trabaja a nivel **análisis estático**:
no ejecuta instrucciones, no evalúa expresiones ni genera código máquina.
Su propósito es **validar que un archivo de entrada esté correctamente escrito según las reglas del lenguaje MIO**.

Este manual explica desde cero qué es el proyecto, cómo está estructurado, cómo se utiliza, y cómo interpretar los casos de uso incluidos.
Está diseñado para que cualquier usuario —incluso sin conocimientos previos sobre compiladores o teoría de lenguajes— pueda comprender exactamente cómo funciona el sistema y qué puede esperar de él.

2. ¿Qué es el Compilador MIO?

El compilador MIO es un conjunto de herramientas que permiten analizar programas escritos en el lenguaje MIO a través de dos etapas fundamentales:

1. **Análisis Léxico:**
   Reconoce palabras, identificadores, operadores y símbolos válidos, y los transforma en una secuencia de tokens.
2. **Análisis Sintáctico:**
   Toma esos tokens y verifica que la estructura del programa cumpla con la gramática formal del lenguaje.

En otras palabras, el compilador responde dos preguntas:

- **¿Todas las palabras escritas son válidas?**
- **¿Las sentencias están ordenadas y escritas de forma correcta según el lenguaje?**

Si ambas preguntas tienen respuesta afirmativa, entonces el programa MIO es válido.

3. Componentes del Proyecto

El proyecto está compuesto por tres módulos principales, cada uno cumpliendo una función específica dentro del proceso de compilación. El usuario no necesita conocer la implementación interna, pero sí comprender **para qué sirve cada archivo** y **cuándo se debe ejecutar**.

3.1 Analizador Léxico — analex.py

Este módulo es el responsable de abrir un archivo con extensión .mio y examinarlo carácter por carácter para identificar:

- Palabras reservadas del lenguaje (como PROGRAMA, SI, MIENTRAS, IMPRIME, etc.)
- Identificadores válidos (variables creadas por el usuario)
- Valores numéricos (hexadecimales)
- Literales de texto
- Operadores aritméticos y relacionales
- El símbolo de asignación =

Durante este proceso también detecta **errores léxicos**, tales como:

- Caracteres no permitidos
- Literales de texto sin cerrar
- Identificadores demasiado largos
- Hexadecimales incorrectos

Si el archivo no presenta errores de este tipo, el analizador genera dos archivos de salida:

- Un archivo .lex, que contiene la secuencia de tokens del programa.
- Un archivo .sim, que funciona como tabla de símbolos, registrando identificadores, literales de texto y valores numéricos encontrados.

3.2 Analizador Sintáctico — anasin.py

Este módulo recibe el archivo .lex generado por el analizador léxico.
Su función es verificar si la secuencia de tokens respeta la **gramática del lenguaje MIO**, es decir, si el orden en que aparecen las sentencias corresponde a un programa bien formado.

El analizador sintáctico valida aspectos como:

- Presencia de la estructura inicial obligatoria:

PROGRAMA nombre
...
FINPROG

- Sentencias bien formadas:
    - Asignaciones
    - Condicionales (SI…ENTONCES…SINO…FINSI)
    - Bucles (MIENTRAS…HACER…FINM)
    - Impresiones y lecturas
- Correcto uso de operadores relacionales y aritméticos
- Expresiones internas válidas dentro de las sentencias

Si el archivo respeta completamente la gramática, la salida será:

Compilación exitosa

En caso contrario, el analizador mostrará el error y la posición exacta del token donde ocurrió el problema.

3.3 Ejecutor de Pruebas — run_tests.py

Este módulo automatiza la ejecución del compilador sobre un conjunto de carpetas de prueba.
Cada carpeta contiene un archivo .mio que se procesa automáticamente. Para cada prueba se generan:

- .lex
- .sim
- salida.log con los resultados del análisis

Este ejecutor permite validar de forma rápida y sistemática el comportamiento del compilador.

4. ¿Cómo usar el compilador?

Aunque el proyecto está compuesto por varios módulos, su uso es sencillo.
A continuación se explica el proceso de manera clara y entendible para cualquier usuario.

4.1 Analizar un archivo MIO

Supongamos que el archivo se llama:

programa.mio

Para hacer el análisis léxico, se utiliza:

python analex.py programa.mio

Si el programa no tiene errores léxicos, se generarán:

programa.lex
programa.sim

A continuación, para el análisis sintáctico:

python anasin.py programa.lex

Si la estructura del programa es válida, se mostrará:

Compilación exitosa

Si existe una sentencia mal formada, el sistema indicará la posición exacta donde ocurrió el error.

5. Casos de Uso del Compilador (Casos 0–5)

El proyecto incluye **seis casos de prueba**, cada uno diseñado para evaluar un aspecto particular del compilador.
A continuación se presenta una explicación detallada, académica y clara de cada caso, por qué fue creado, qué espera validar y qué comportamiento demuestra del compilador.

Caso 0 — Prueba de Funcionamiento General

El caso 0 es el punto de partida del compilador. Está diseñado como un programa simple cuya estructura es válida y que permite verificar que ambos analizadores —léxico y sintáctico— tengan un comportamiento adecuado.

Este caso típicamente incluye:

- Un encabezado PROGRAMA
- Un identificador principal
- Sentencias simples como asignaciones o impresiones

- Un final de programa FINPROG

Su propósito es asegurar que:

- El analizador léxico reconoce correctamente los tokens básicos.
- El analizador sintáctico entiende correctamente la estructura mínima de un programa MIO.
- No aparecen errores inesperados en la cadena de operación inicial.

El usuario puede correr este caso para verificar que el entorno esté configurado correctamente.

Caso 1 — Evaluación de Asignaciones y Expresiones Básicas

El caso 1 está diseñado para comprobar el manejo de sentencias de **asignación** y el reconocimiento de **expresiones aritméticas simples**. Esto es fundamental para comprobar que el analizador léxico reconoce correctamente:

- Identificadores
- Valores hexadecimales
- Operadores aritméticos
- El operador de asignación

Y que el analizador sintáctico permita expresiones de la forma:

[id] = [id] [op_ar] [val]

Además, este caso evalúa si el compilador detecta errores comunes como:

- Asignaciones incompletas
- Operadores faltantes
- Valores mal escritos

Un usuario que desea comprender cómo maneja el compilador expresiones puede estudiar este caso para observar el proceso completo desde .mio hasta .lex y .sim.

Caso 2 — Condicionales y Comparaciones Relacionales

Este caso introduce estructuras de control mediante **sentencias condicionales**.
Su propósito principal es validar dos niveles de comportamiento:

1. El reconocimiento léxico de operadores relacionales (<, >, <=, >=, <>, ==).

2. La validación sintáctica de comparaciones y bloques anidados.

El caso incluye sentencias como:

```
SI x < 5 ENTONCES
   ...
SINO
   ...
FINSI
```

Este caso es determinante porque permite observar cómo el compilador maneja estructuras que requieren emparejamiento correcto de palabras reservadas: ENTONCES, SINO y FINSI.

Un error común que este caso ayuda a detectar son bloques sin cerrar o fuera de orden.

Caso 3 — Entrada, Salida y Manejo de Literales de Texto

El caso 3 incorpora instrucciones de **entrada y salida** mediante las sentencias:

- IMPRIME
- LEE

También introduce **literales de texto**, los cuales requieren un manejo especial en el análisis léxico debido a:

- Comillas obligatorias
- Longitud de la cadena
- Identificación del final del literal

Este caso valida:

- Que el analizador léxico distinga correctamente entre expresiones y texto.
- Que el analizador sintáctico acepte las dos variantes de IMPRIME:

```
IMPRIME [txt]
IMPRIME <EXPR>
```

El usuario puede observar cómo se registran los textos detectados en la tabla de símbolos .sim.

Caso 4 — Manejo de Errores Léxicos y Sintácticos

El caso 4 es un caso deliberadamente incorrecto creado para probar la robustez del compilador.
Este archivo .mio incluye errores intencionales como:

- Tokens inesperados
- Sentencias mal formadas
- Literales sin cerrar
- Palabras reservadas mal ubicadas

Este caso es fundamental desde la perspectiva académica porque demuestra que el compilador:

- No solo valida lo correcto
- Sino que también **detecta y reporta lo incorrecto** con precisión

El archivo salida.log generado muestra de forma clara los errores, permitiendo estudiar cómo el compilador maneja fallas.

Caso 5 — Caso de Integración Completa

Este es el caso más completo del proyecto. Incluye:

- Asignaciones
- Condicionales
- Bucles
- Expresiones aritméticas
- Entradas y salidas
- Literales de texto

Este caso funciona como:

- Validación final del compilador
- Demostración del funcionamiento integral
- Caso ideal para presentar durante una exposición o evaluación

El usuario puede observar cómo se genera un .lex amplio, una tabla .sim completa y una salida sintáctica exitosa.

6. Interpretación de Resultados

El usuario puede esperar dos tipos de resultados:

Resultados Léxicos

El archivo .lex permite verificar si todas las palabras del programa fueron reconocidas.
El .sim muestra los identificadores, textos y valores usados.

Resultados Sintácticos

Si la sintaxis es correcta:

Compilación exitosa

Si hay un error:

- Se indica el token encontrado
- El token esperado
- La posición exacta dentro del programa

Esto permite corregir rápidamente errores en el archivo .mio.