TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica
Escuela de ingeniería en computación

IC-5701 Compiladores e Intérpretes
PY02 - Análisis Sintáctico

Estudiante(s):

Joselyn Priscilla Jiménez Salgado	2021022576
Dylan Montiel Zúñiga	2023205654

Profesor(a) a cargo:

Allan Rodríguez Dávila

Verano 2024-2025

Tabla de contenidos

Manual de Usuario

Este manual proporciona las instrucciones necesarias para compilar el proyecto, ejecutar y usar el analizador lexico con jflex y cup. Este proyecto fue desarrollado con la herramienta de desarrollo Apache NetBeans.

Configuración del entorno

1. **Instalación de la herramienta de desarrollo y dependencias**

Si no tienes NetBeans instalado, descárgalo desde la página oficial e instálalo en tu equipo. Asegúrate de instalar las herramientas necesarias (por ejemplo, soporte para Java el cual lo puede encontrar en Oracle JDK).

2. **Crear un directorio para el proyecto (opcional)**

```
mkdir AnalizadorLexico
```

3. **Clonar el repositorio (en el cmd)**

```
git clone https://github.com/MITTuu/PP02-Compiladores_E_Interpretes.git
```

4. **Cargar el proyecto con NetBeans**

a) Ve al menú principal y selecciona:

File > Open Project (Archivo > Abrir Proyecto).

b) Navega hasta el directorio donde se encuentra el proyecto.

c) Selecciona la carpeta “programa”.

d) Una vez hecho esto, revisa la ventana de proyectos en NetBeans para asegurarte de que todos los archivos y dependencias están disponibles.

Imagen 1: Abrir el proyecto

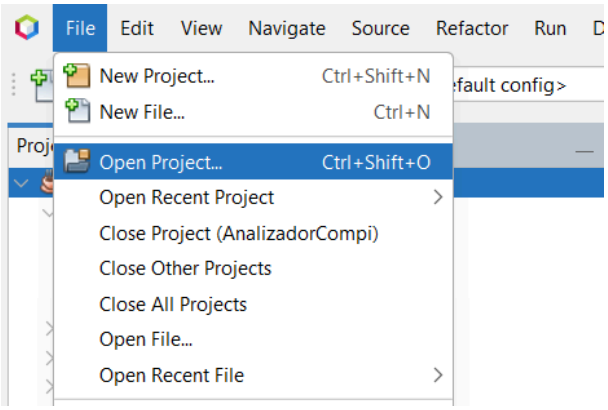


Imagen 2: Seleccionar el directorio programa

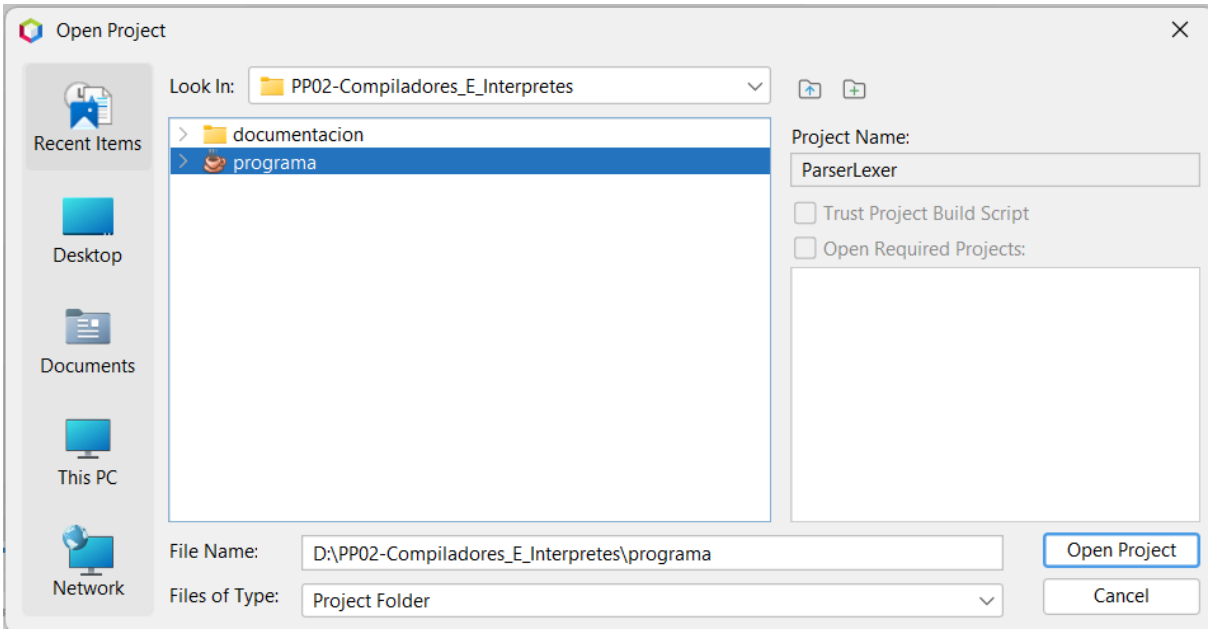
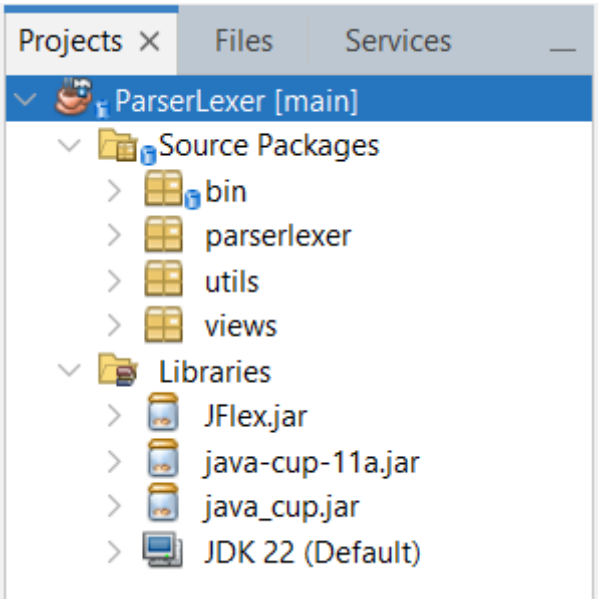


Imagen 3: Proyecto cargado correctamente



1. Instrucciones de compilación y ejecución

a. Generación de Archivos Autogenerados (JFlex y CUP)

Aunque los archivos autogenerados ya están disponibles en el directorio *src/bin/*, se recomienda regenerarlos para garantizar el correcto funcionamiento del programa y evitar posibles errores.

Para generar los archivos de código fuente (sym.java, parser.java y LexerCup.java) utilizados en el análisis léxico y sintáctico, es necesario compilar y ejecutar la clase principal Main.java, ubicada en el directorio *src/utils/*.

Imagen 4: Compilar la clase Main.java

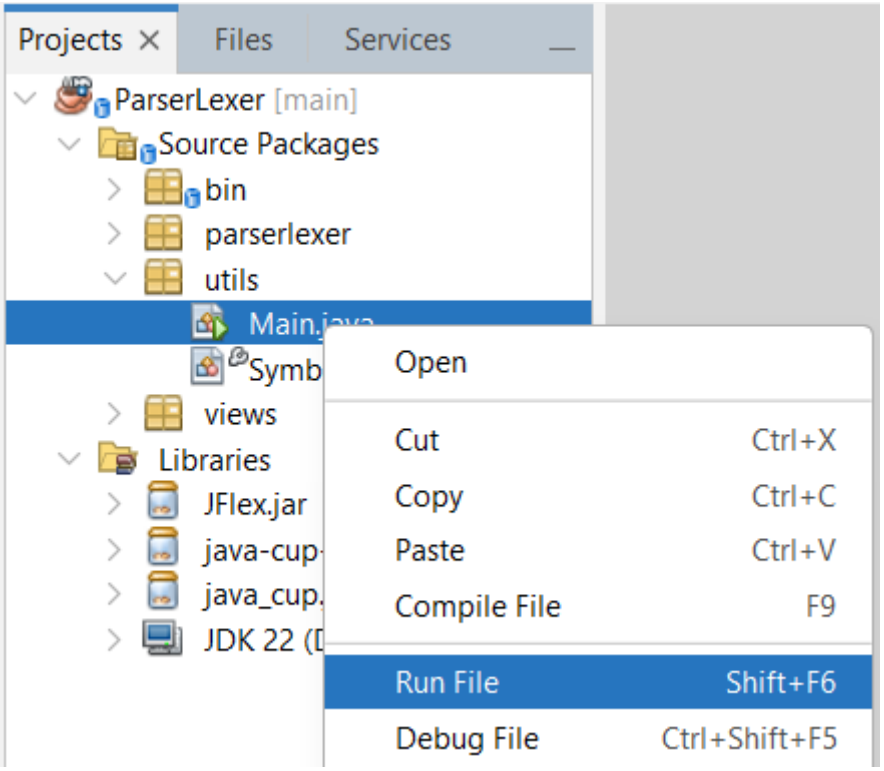
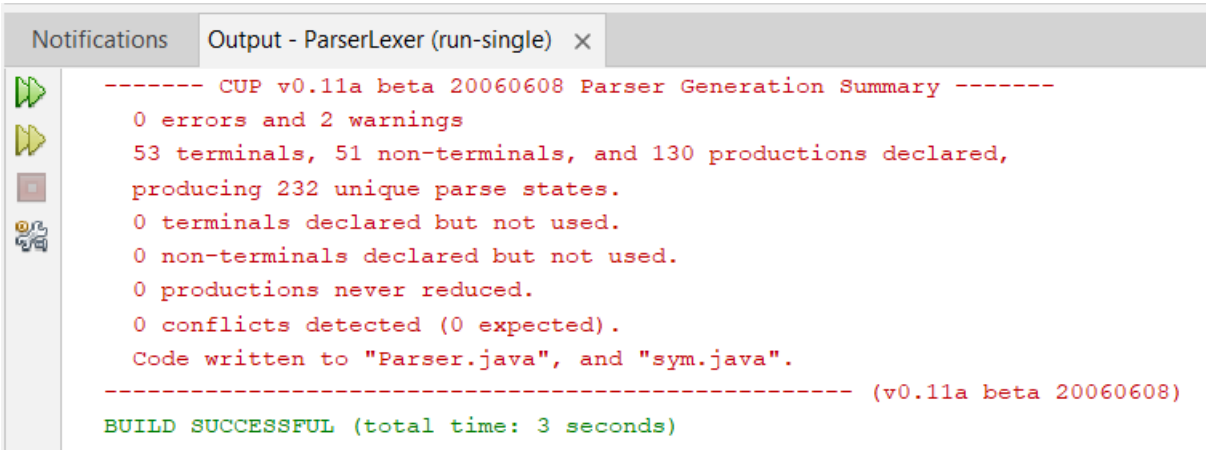


Imagen 5: Verificar la salida en la terminal



b. Probar el análisis léxico y sintáctico

Para probar el análisis léxico y sintáctico utilizando el archivo generado por JFlex (Lexer.java, Parser.java y sym.java), el proyecto cuenta con la interfaz gráfica View_Main.java, ubicada en el directorio *src/views/*. Esta interfaz permite cargar archivos de entrada y visualizar los resultados del análisis léxico y sintáctico de manera interactiva, facilitando el uso y la comprensión del funcionamiento del programa, y si se desea se puede guardar el análisis generado en un archivo TXT.

Imagen 6: Compilar y ejecutar la interfaz para probar el analizador léxico

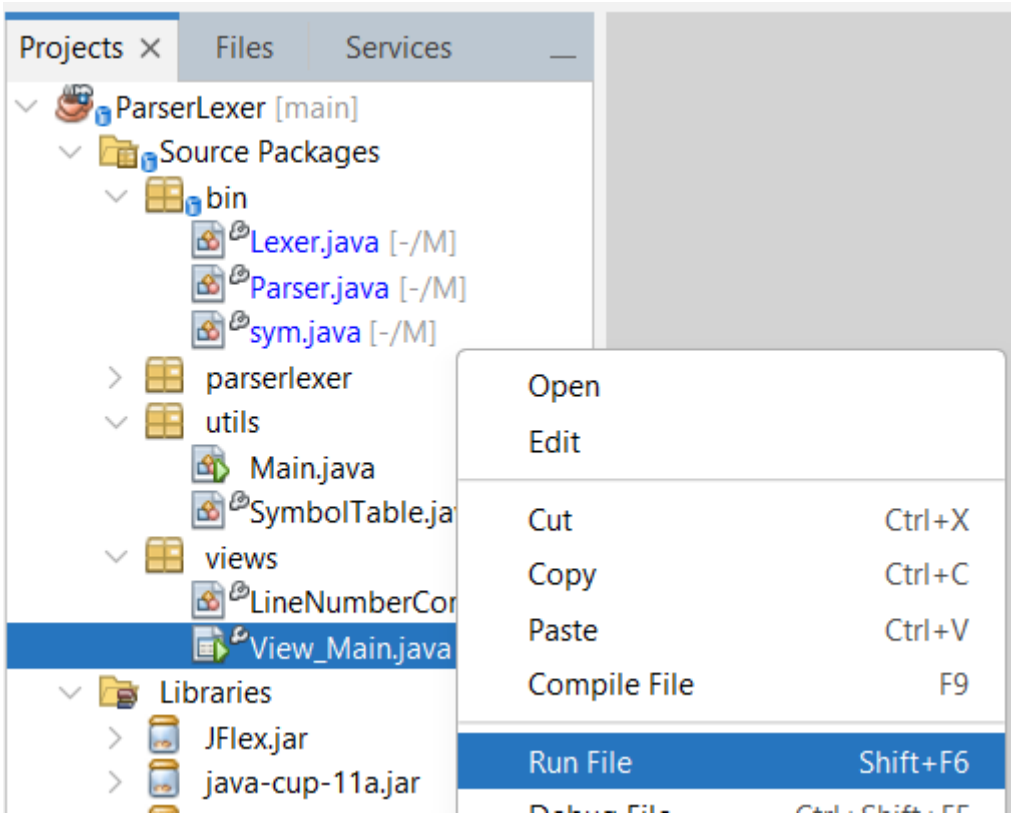
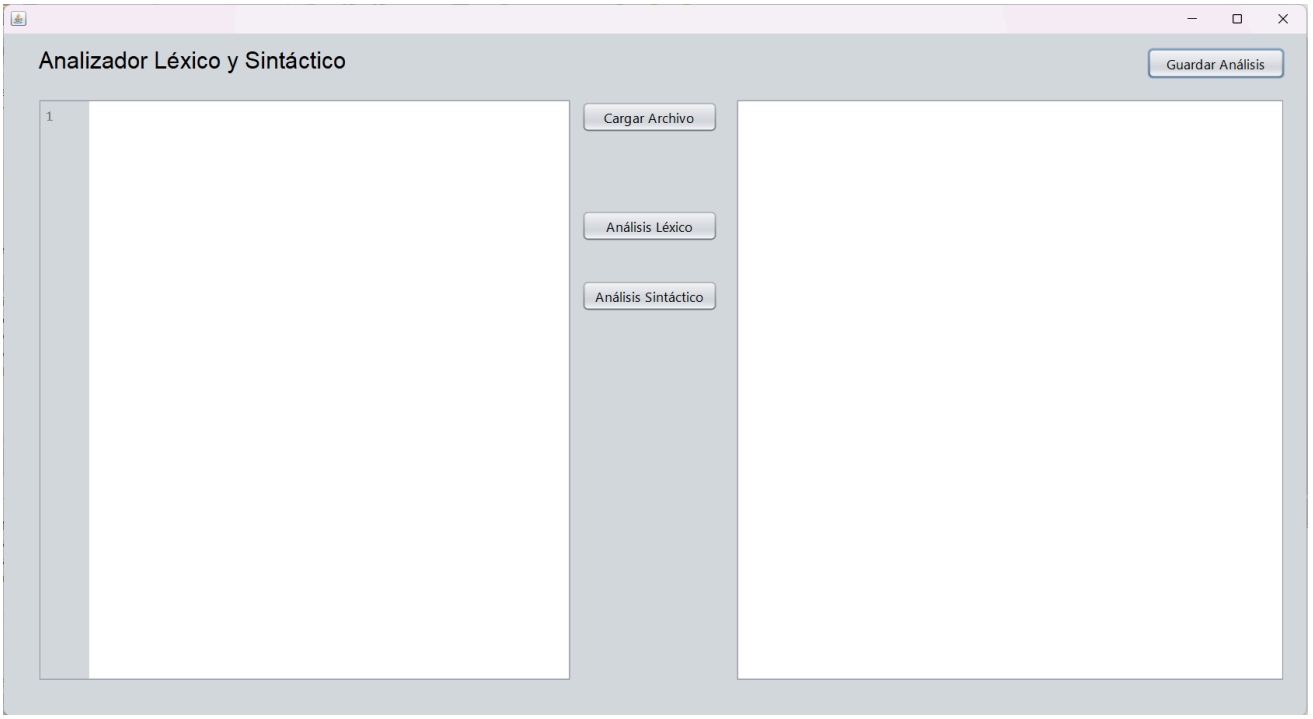


Imagen 7: Interfaz

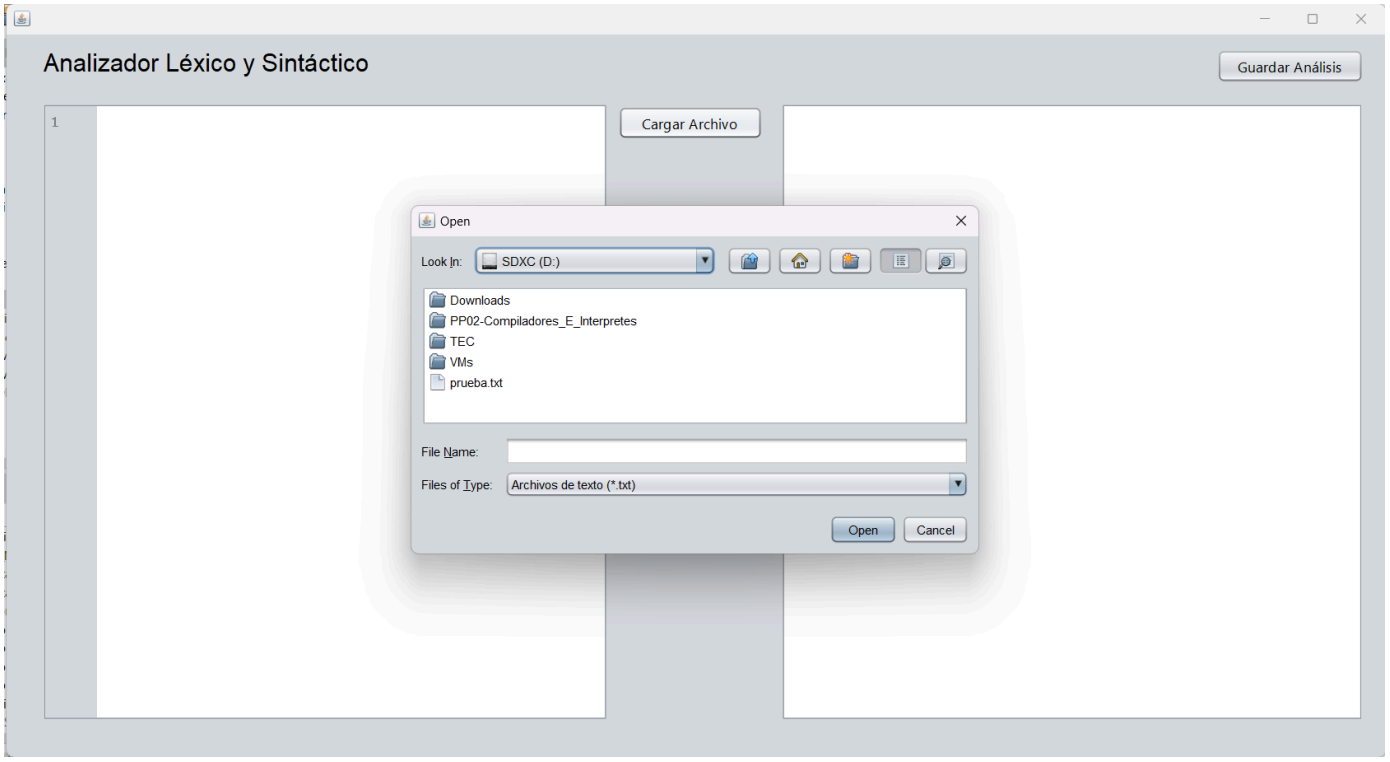


La interfaz gráfica es muy sencilla, se puede cargar archivos TXT para analizar o se puede escribir directamente en el cuadro de la izquierda, los botones para generar análisis léxicos y sintácticos muestran el resultado en el cuadro de la derecha. En el siguiente apartado se muestra a detalle su uso completo.

Pruebas de funcionalidad

La interfaz es muy sencilla, se divide en dos secciones (parte izquierda y parte derecha). Entre estas partes encontramos el botón para cargar un archivo de texto para su posterior análisis y también encontramos dos botones para hacer los análisis correspondientes. En el panel de texto izquierdo (habilitado para su escritura) se puede escribir y/o mostrar la información cargada. En la parte derecha encontramos el botón para guardar el análisis realizado, al generar un análisis su resultado se muestra en el panel de texto derecho (no habilitado para su escritura).

Imagen 8: cargar archivo de texto plano



Al tocar el botón ‘Cargar Archivo’ se muestra un buscador de archivos donde se deberá seleccionar un archivo de texto plano con el código a analizar.

Imagen 9: Contenido cargado

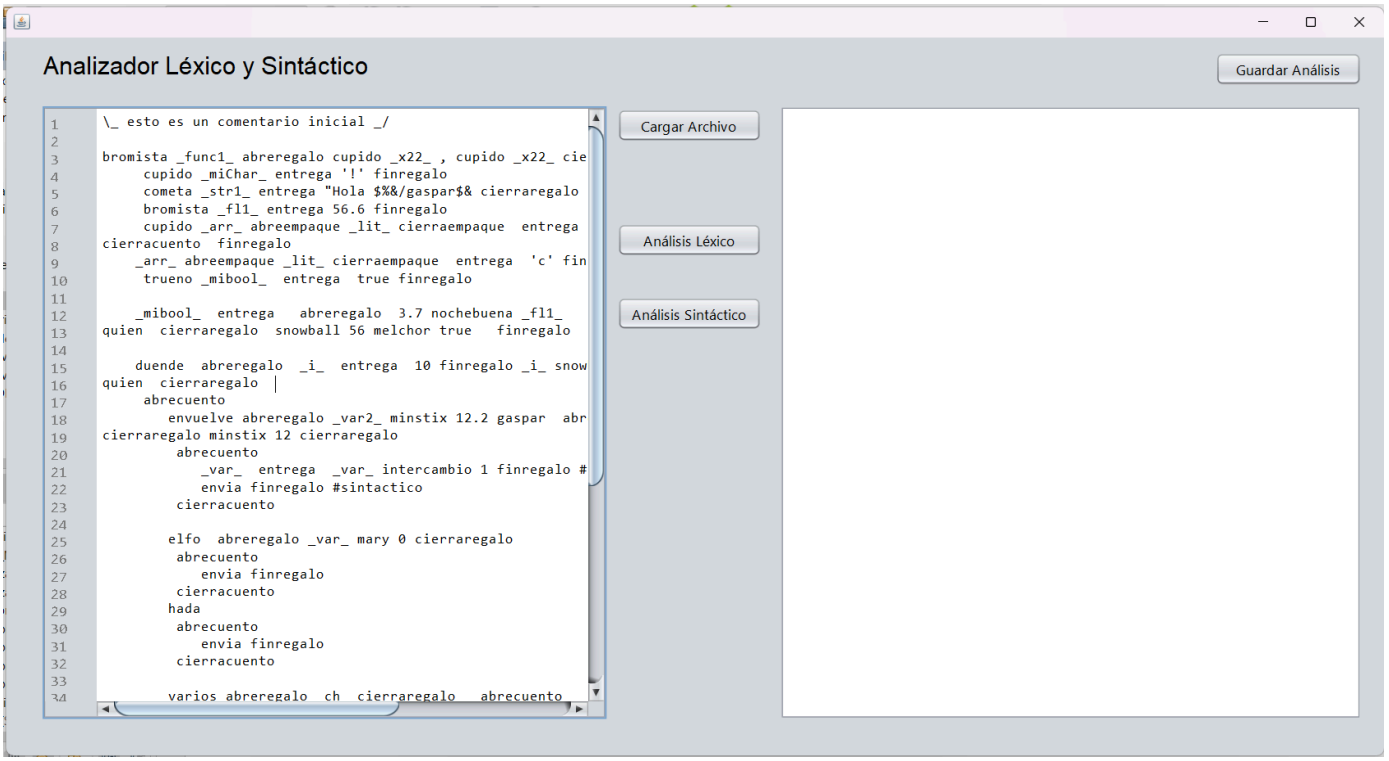
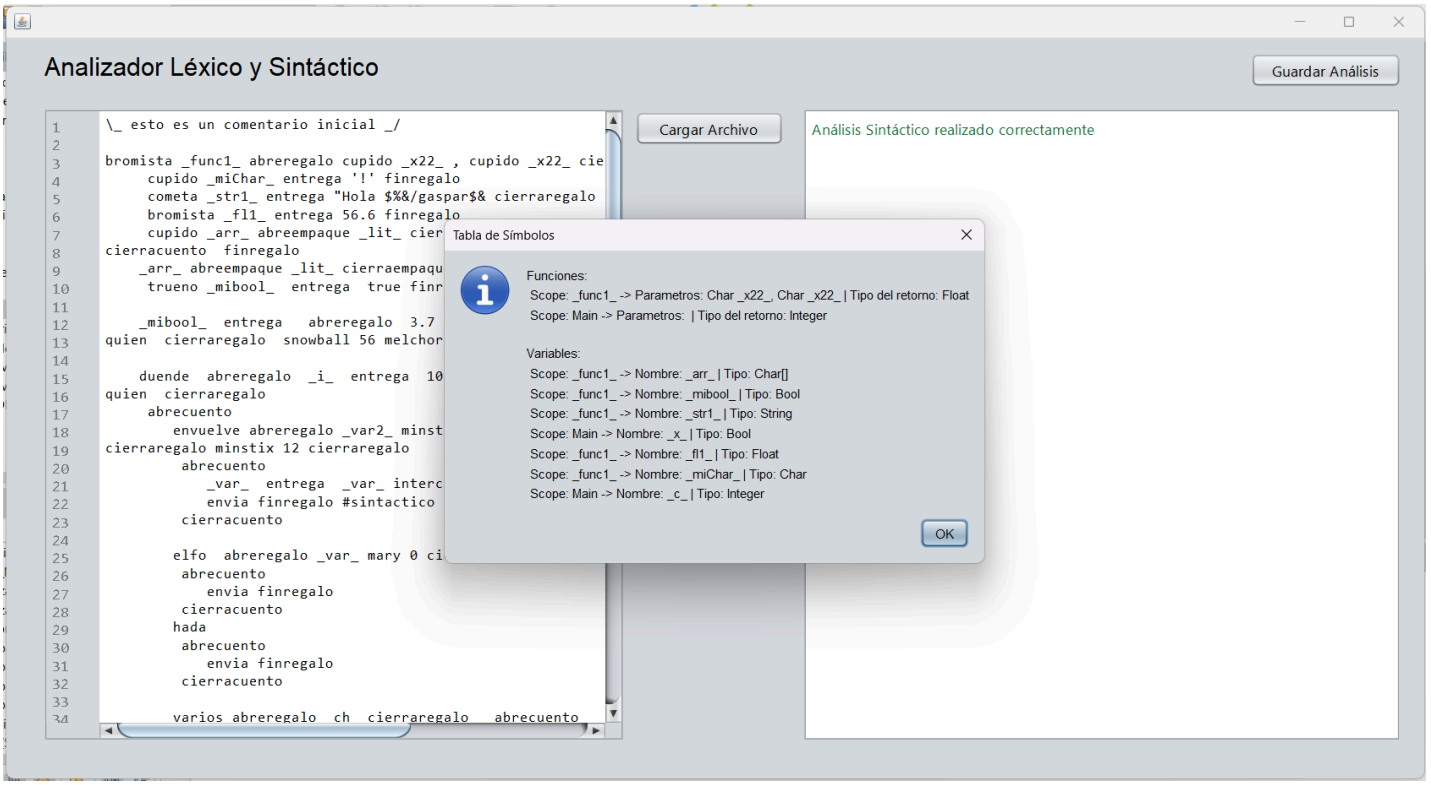
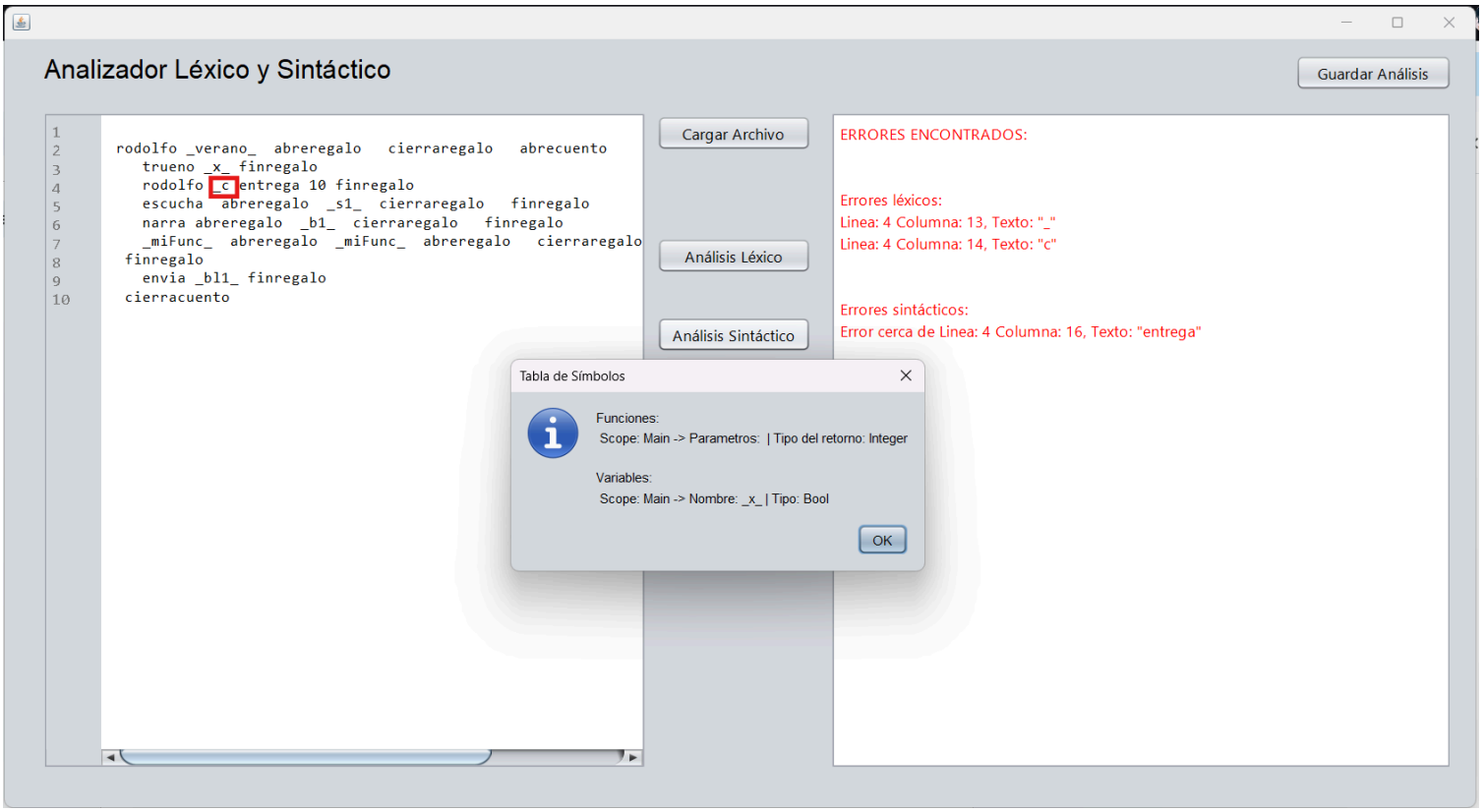


Imagen 10: Análisis sintáctico



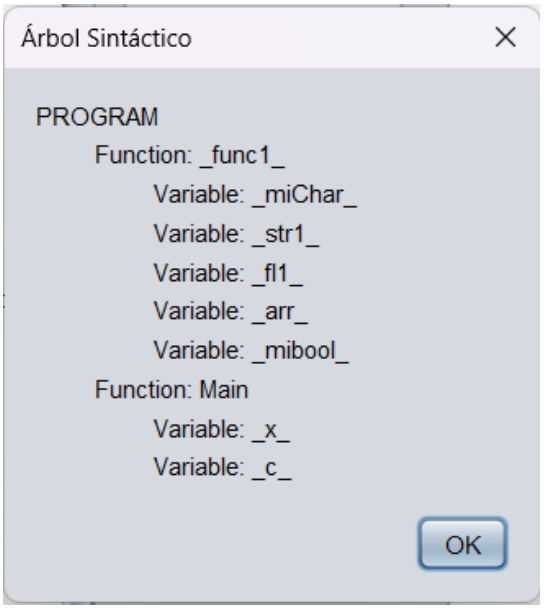
Al generar el análisis sintáctico, se muestra un pop-up con la información de la tabla de símbolos y en el panel de texto derecho indica si hubo algún error durante el análisis.

Imagen 11: manejo de errores léxicos



En caso de que el programa encuentre un error léxico durante el análisis, si el error no es crítico, el análisis se concluye generando la tabla de símbolos obviando el error encontrado como se muestra en la imagen.

Imagen 12: Árbol Sintáctico



El árbol sintáctico fue la última funcionalidad desarrollada en el proyecto y aún se encuentra en una etapa inicial. Actualmente, muestra de manera jerárquica las funciones del programa junto con sus variables. Debido al tiempo que se dispuso para su desarrollo, esta es la única información que se logró implementar en esta etapa, refiérase a la imagen 12 para ver un ejemplo.

Descripción del problema

En este proyecto, se busca diseñar un analizador sintáctico para un lenguaje imperativo destinado a configuraciones de hardware, con soporte para variables, funciones y estructuras de control básicas. La gramática del lenguaje debe ser procesada por el analizador para identificar y reportar errores de manera eficiente. Esto es esencial para evitar que errores sintácticos se propaguen a las fases posteriores del desarrollo, como la generación de código o la ejecución, lo cual podría comprometer la funcionalidad de los sistemas empotrados.

El problema radica en implementar un sistema capaz de interpretar correctamente las reglas definidas por la gramática y procesar los programas de entrada, devolviendo mensajes claros y detallados en caso de errores. Además, se requiere que el analizador sea extensible y eficiente para adaptarse a posibles futuras modificaciones del lenguaje.

Diseño del programa

El analizador sintáctico es responsable de verificar que la secuencia de tokens generada por el analizador léxico cumple con las reglas gramaticales definidas para el lenguaje C. Para ello, se utiliza **CUP**, una herramienta que permite definir una gramática formal en un archivo .cup y generar código Java que implementa un parser basado en esta gramática.

El diseño se centra en una estructura modular que combina el análisis léxico y sintáctico para identificar errores de forma temprana en el proceso de compilación. Primero, el analizador léxico, implementado con **JFlex**, genera una lista de tokens clasificados según su tipo y ubicación. Estos tokens son luego enviados al analizador sintáctico, que utiliza un enfoque basado en el análisis LR (Left-to-right, Rightmost derivation) para validar su orden y estructura.

La gramática definida en el archivo .cup describe las reglas de formación del lenguaje, incluyendo las producciones para estructuras como declaraciones de variables,

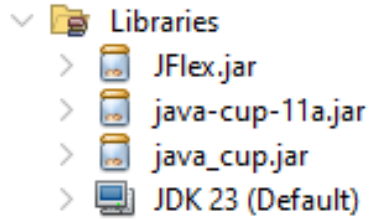
instrucciones condicionales, bucles, funciones y operadores. Durante el análisis, el parser construye un árbol de derivación que representa la estructura jerárquica del código fuente. Si se detecta un error sintáctico, se genera un mensaje detallado que indica el tipo de error y su ubicación, facilitando la corrección.

Además, el analizador sintáctico se integra con estructuras de datos intermedias que pueden ser utilizadas para futuras etapas, como la generación de código intermedio o la optimización. Este diseño modular permite la escalabilidad del programa, permitiendo la adición de nuevas reglas gramaticales o ajustes en la validación.

En el proyecto, el analizador sintáctico cumple un papel crucial para garantizar que el código fuente analizado no solo sea válido léxicamente, sino que también respete las reglas de construcción del lenguaje C, proporcionando así una base sólida para análisis posteriores.

Librerías usadas

Las librerías usadas que se necesitaron para la creación de este proyecto:



1. **JFlex.jar**
    - **Uso:**

      JFlex es una herramienta para generar analizadores léxicos. Permite definir un archivo de especificación que contiene las expresiones regulares y las acciones asociadas para reconocer los tokens de un lenguaje. A partir de este archivo, JFlex genera un analizador léxico en Java, que puede integrarse con otras herramientas para análisis sintáctico.
    - **Aplicación en el proyecto:**

      En este proyecto, JFlex se utiliza para construir el componente de análisis léxico, encargado de identificar y clasificar las unidades léxicas del código fuente escrito en el lenguaje definido. Esto incluye palabras clave, identificadores, literales y operadores, garantizando que cada token sea reconocido de acuerdo con las reglas definidas en la gramática del lenguaje.

2. **java-cup-11a.jar / java_cup.jar**
    - **Uso:**

      Java CUP es una herramienta para generar analizadores sintácticos en Java. CUP toma como entrada un archivo de especificación que define la gramática del lenguaje y las acciones semánticas correspondientes, produciendo como salida un parser en Java.
    - **Aplicación en el proyecto:**

      En este proyecto, CUP se emplea para implementar el analizador sintáctico. Este componente verifica que la secuencia de tokens generada por el analizador léxico cumpla con las reglas gramaticales del lenguaje. Además, permite realizar acciones semánticas para preparar la estructura del programa (por ejemplo, la generación de árboles de sintaxis abstracta) que se utilizará en las etapas posteriores del compilador.

Análisis de resultados

Objetivos Alcanzados

- Implementación de un Analizador Sintáctico Funcional:
  - Se diseñó e implementó un analizador sintáctico que procesa correctamente programas escritos en el lenguaje definido.
- Validación de Estructuras Gramaticales:
  - Se comprobó que el analizador valida correctamente declaraciones de variables, estructuras de control (**if, while, for**), funciones y expresiones aritméticas.
- Detección de Errores Sintácticos:
  - El sistema detecta y reporta errores sintácticos con mensajes claros, especificando el tipo de error.
- Generación de Árboles de Derivación:
  - El analizador construye correctamente árboles de derivación que representan la estructura jerárquica de los programas de entrada.
- Pruebas de Robustez:
  - El sistema manejó con éxito programas con múltiples errores y probó ser capaz de continuar el análisis después de errores parciales.
- Modularidad y Extensibilidad:
  - El diseño modular permite la integración con futuras fases del compilador, como el análisis semántico y la generación de código intermedio.

Objetivos no alcanzados

Todos los resultados se lograron dentro del marco de tiempo y los recursos disponibles. Los problemas o desafíos que surgieron durante el desarrollo fueron resueltos a tiempo, y no hubo impedimentos que impidieran alcanzar los objetivos planteados.

Bitácora

Link del repositorio: https://github.com/MITTuu/PP02-Compiladores_E_Interpretes.git