INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

CENTRO ACADÉMICO DE LIMÓN

PROYECTO 2

PROF. Allan Rodríguez Dávila

ESTUDIANTE: Alex Sánchez Céspedes

CARNÉ: 2018014778

SEMESTRE I 2023
Puerto Limón, Costa Rica

1. DESCRIPCIÓN DEL PROBLEMA

Se desea diseñar un nuevo lenguaje de programación de paradigma imperativo con una gramática liviana y que permita realizar operaciones básicas para la configuración de chips. Este lenguaje es totalmente esencial, ya que la industria de chips está en constante crecimiento y requiere lenguajes de programación cada vez más potentes y livianos.

En implementaciones anteriores se añadieron funcionalidades para el analizador léxico y sintáctico de dicho lenguaje, en donde el analizador léxico se encargará de reconocer y clasificar los diferentes componentes léxicos presentes en el código fuente, como identificadores, números, operadores y palabras reservadas, mientras que el analizador sintáctico se encargará de verificar la estructura sintáctica del programa para asegurarse de que cumpla con la gramática definida para el lenguaje.

En este documento se presenta una gran mejora al analizador léxico y sintáctico, además de las funcionalidades del analizador semántico y la generación de código intermedio.

2. DISEÑO DEL PROGRAMA

Para el desarrollo tanto del analizador léxico como el sintáctico se utilizaron varias herramientas y librerías desarrolladas en lenguaje JAVA, para el analizador léxico se utilizó la herramienta JFlex y para el analizador sintáctico se utilizó la herramienta CUP. Estas herramientas son utilizadas desde la primera implementación del compilador del lenguaje y con esta implementación del análisis semántico y generación de código intermedio se pretende mantener la continuidad en el aprovechamiento de estas, además se desarrollaron algunas clases y funciones adicionales en lenguaje JAVA para facilitar las validaciones y verificaciones del proceso semántico, estas funcionalidades se detallarán en la siguiente sección.

Para el desarrollo del análisis semántico, se añadió la funcionalidad requerida en cada una de las producciones con las validaciones e impresión de mensajes informativos al usuario.

3. LIBRERÍAS UTILIZADAS

```
java.io.BufferedReader
java.io.BufferedWriter
java.io.File
java.io.FileNotFoundException
java.io.FileReader
java.io.FileWriter
java.io.IOException
java.io.Reader
java.nio.file.Files
java.nio.file.Paths
JFlex
java_cup.runtime.*
```

Se creó la clase VerificadorTipos para realizar las validaciones de tipos en el proceso de análisis semántico con las siguientes funcionalidades:

- verificarEntero
- verificarBooleano
- verificarFloat
- verificarChar
- verificarString

4. MANUAL Y PRUEBAS DE FUNCIONALIDAD

La ejecución del proyecto es muy similar a la ejecución de la implementación pasada: Para la ejecución de esta segunda implementación al compilador se utilizará el IDE NetBeans por las facilidades que este presenta para su ejecución; sin embargo, se puede utilizar un IDE de su gusto como Visual Studio Code u otros.

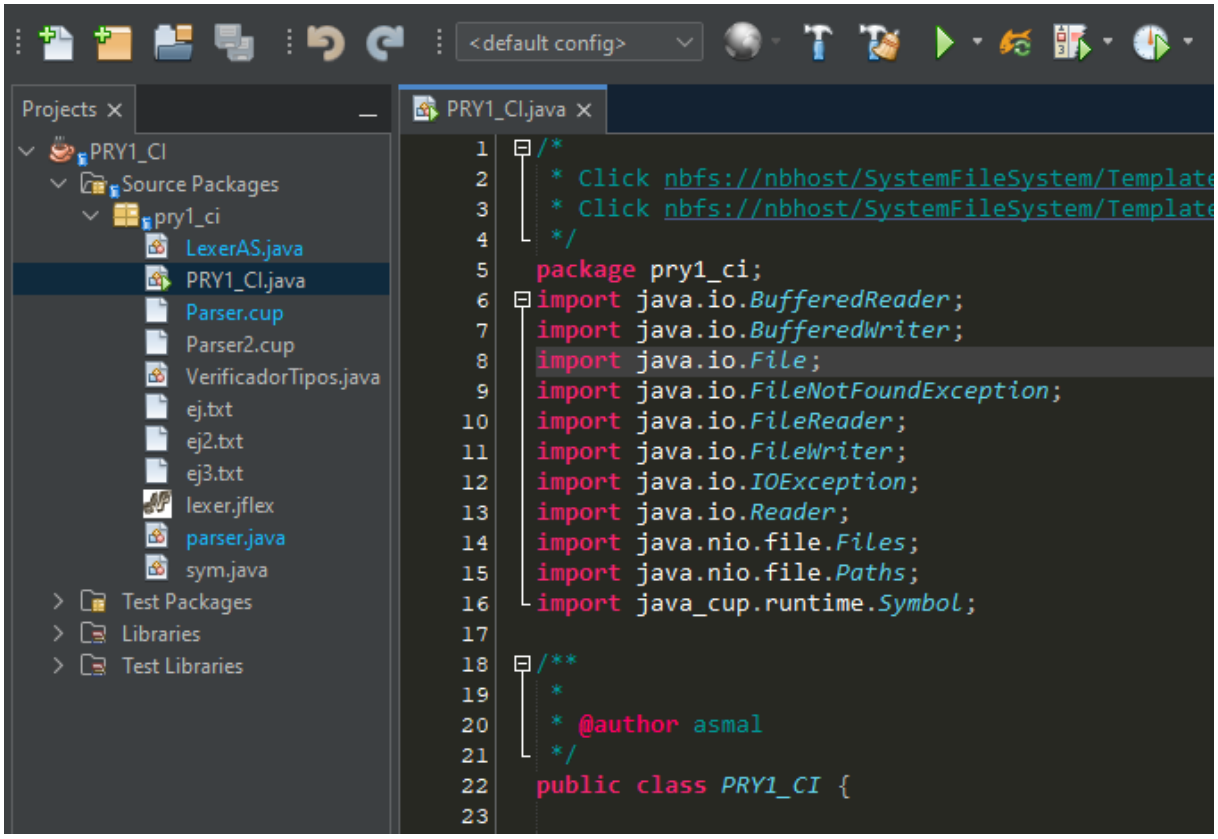

Figura 1: Proyecto abierto desde IDE NetBeans.

Nos ubicaremos en el archivo de nombre **PRY1_CI.java** y lo ejecutaremos dando clic en el botón resaltado, el programa realizará el análisis léxico, sintáctico y semántico del archivo nombrado como **ej.txt**, el cual podremos modificar agregando o eliminando partes del código para realizar nuestras propias pruebas.

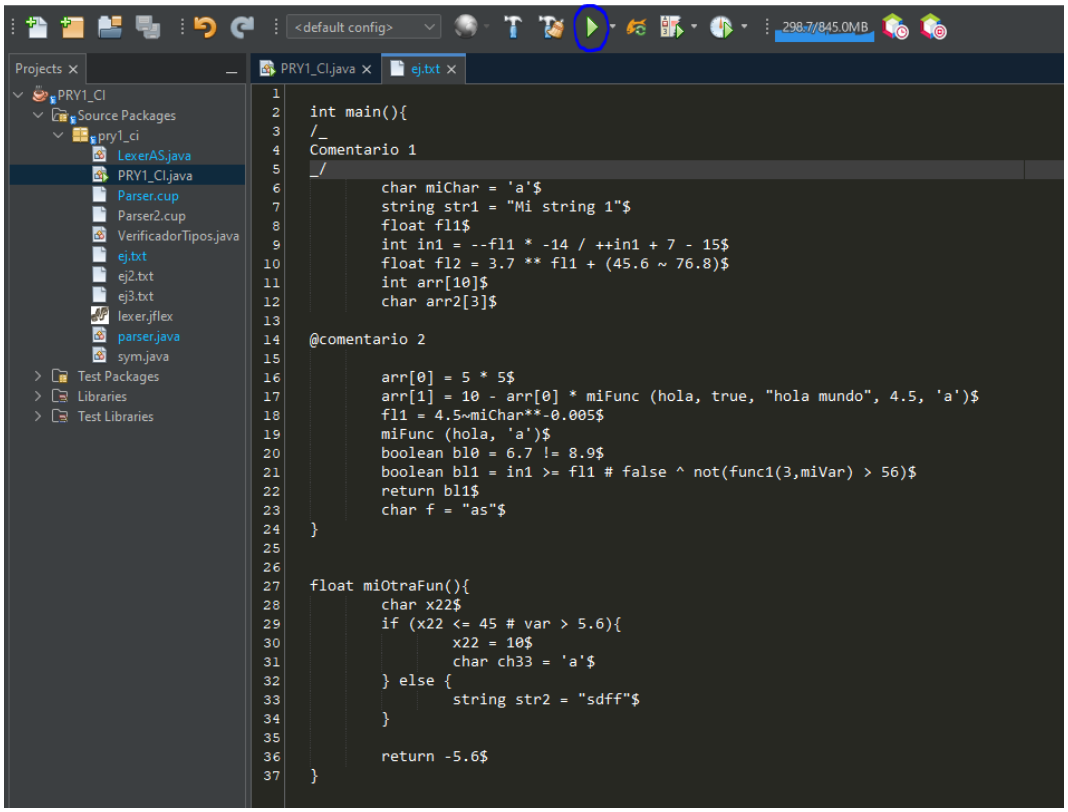

Figura 2: Archivo de prueba ej.txt.

Al ejecutar el programa se nos imprimirá en la consola toda la información de los análisis, si el código fuente presenta errores léxicos, sintácticos o semánticos se presentarán varios mensajes de error con un poco de información del error como por ejemplo la línea de código en donde se encuentra el problema, además se realiza la impresión de la tabla de símbolos con todos los valores e información importante del código.

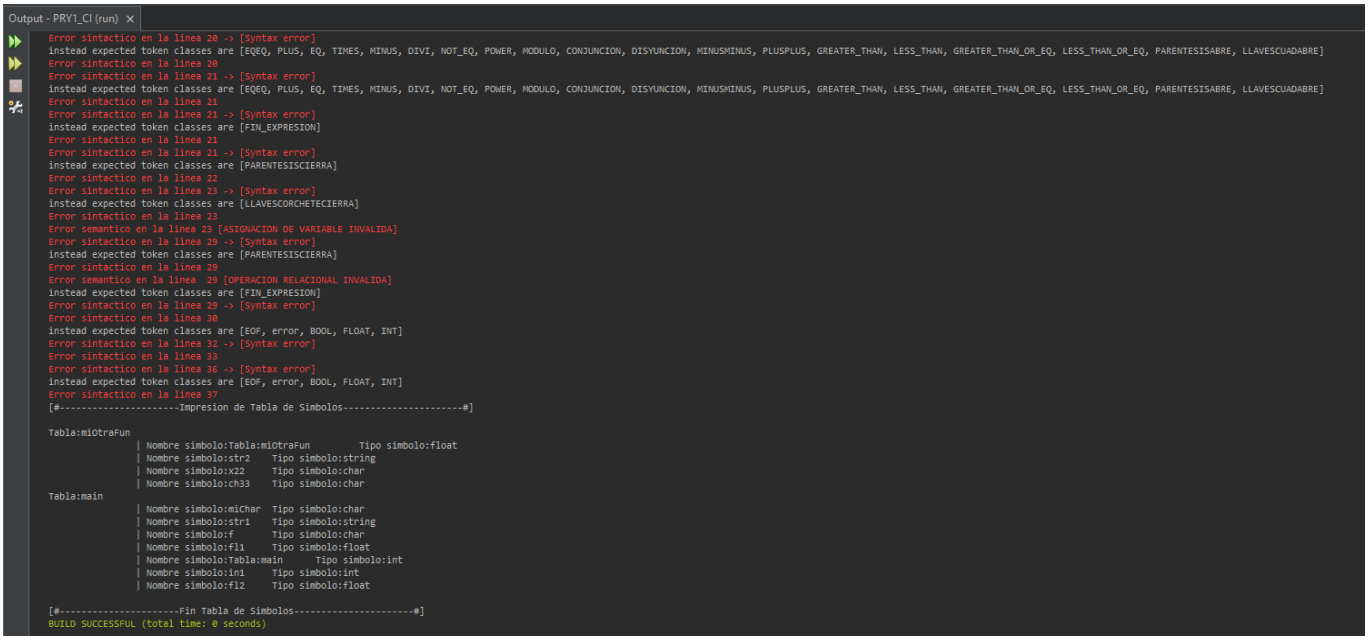

Figura 3: Resultados mostrados en la consola de NetBeans.

5. ANÁLISIS DE RESULTADOS

5.1. LECCIONES APRENDIDAS

Se amplió el conocimiento sobre el funcionamiento de la librería JFLEX y CUP, además de conocer de una manera más amplia el funcionamiento del lexer con el parser al trabajar en conjunto. Se logró aprender sobre la generación de validaciones semánticas y la importancia de la tabla de símbolos para este análisis.

Además, se logró entender la importancia de las reducciones de las instrucciones para la generación de código intermedio.

5.2. OBJETIVOS ALCANZADOS

- Generación de validaciones semánticas.
- Generación de validaciones sintácticas.
- Generación de código intermedio.
- Generación de un parser con amplia funcionalidad para realizar análisis.
- Dominio del tema sobre análisis sintácticos y semánticos.

5.3. OBJETIVOS NO ALCANZADOS

Se plantea que algunos errores sintácticos o semánticos en específico no puedan ser capturados por el parser, además algunos errores sintácticos pueden cambiar el funcionamiento del parser. También, se plantea que la generación de código intermedio no es completa, ya que falto la implementación para la generación de varias partes de las producciones. Las razones de esto es porque al realizar el trabajo una sola persona se complicó el realizar un parser que complemente al 100 % las necesidades del lenguaje, siendo esto más que todo un problema de tiempo para implementación más no de conocimiento.

6. BITÁCORA

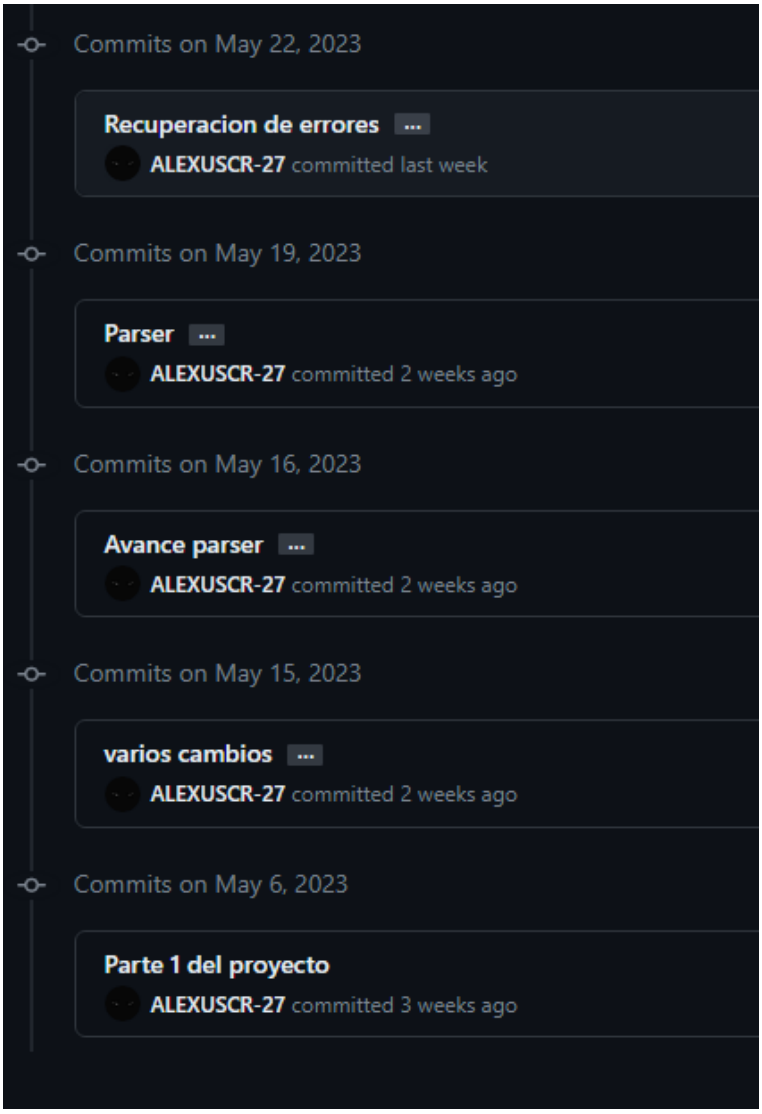

Figura 4: Commits de Github

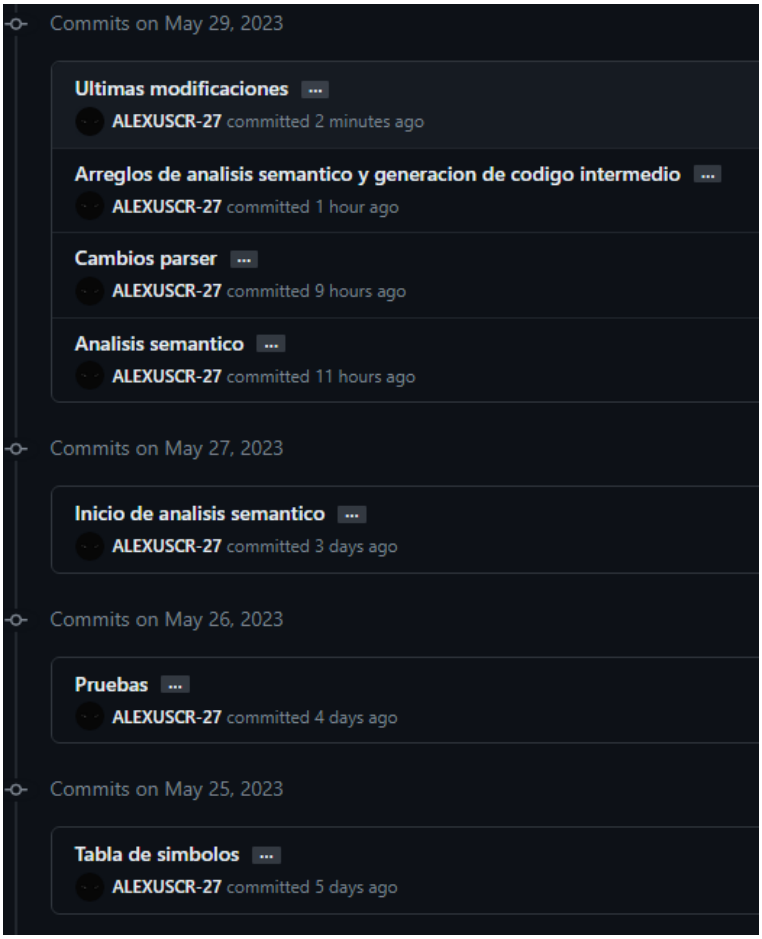

Figura 5: Commits de Github