COMPILADORES E INTERPRETES

PROYECTO 3

ESTUDIANTE:
JORDANO ESCALANTE

PROFESOR:

ALLAN RODRIGUEZ

I SEMESTRE 2025

Manual de usuario: instrucciones de compilación, ejecución y uso bien detalladas.

Requisitos previos

Antes de compilar el proyecto, asegúrese de tener instalado:

- Java Development Kit (JDK) 17 o superior.

- Las siguientes librerías:
- **JFlex** (generador de analizadores léxicos para Java).
- **CUP** (Constructor of Useful Parsers, generador de analizadores sintácticos).
- Java CUP Runtime (para ejecutar el parser generado por CUP)

- **Compilar el proyecto**

  - Ejecute el siguiente comando para generar el analizador sintáctico

    Java -jar src/libs/jflex-full-1.9.1.jar -d src src/BasicLexerCup.jflex

  - Ejecute el siguiente comando para generar el analizador léxico:

    Java -jar src/libs/java-cup-11b.jar -symbols sym -destdir src src/BasicParser.cup

- **Ejecutar el programa**

  - Para compilar el programa utilice el siguiente comando:
    javac -cp "src/libs/*" *src/*.java

  - Para ejecutar el análisis sintáctico y léxico debe ejecutar la clase App.java, asegúrese de ejecutarla desde la línea de comandos para evitar problemas de cash de vs code o su editor:

    Java -cp "src;src/libs/*" App

(Opcional) Limpiar archivos compilados

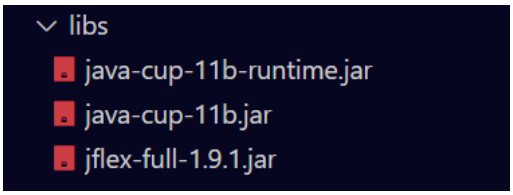
En algunos clasos las clases generadas al compilar pueden ocasionar problemas de compilación, elimínelas y vuelva a crearlas:
Eliminar las clases:
del src*.class

Compilar nuevamente:

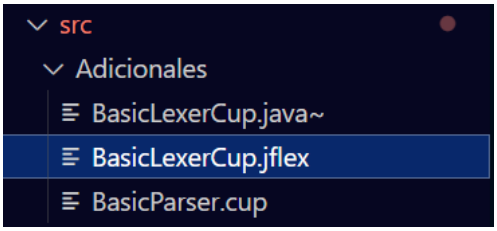
javac -cp "src/libs/*" *src/*.java

Pruebas de funcionalidad: Incluir screenshots.

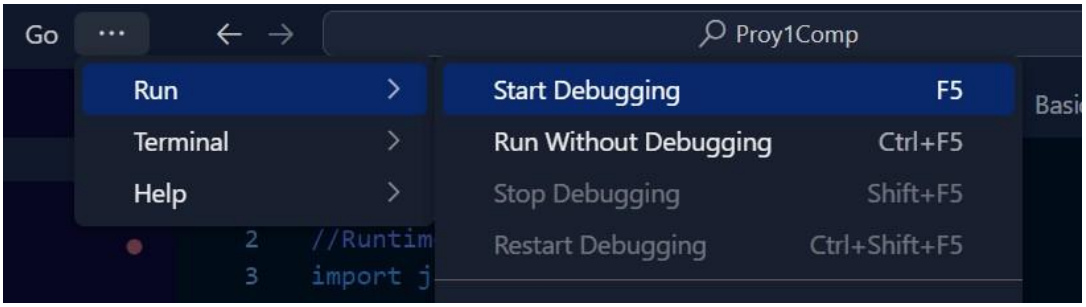
1 Como primer paso se debe tener incluidas las librerías de Jflex y Cup en la carpeta libs.



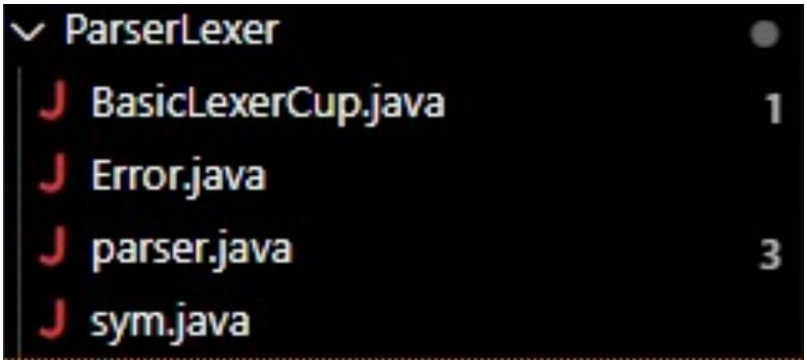
2 Como segundo paso se debe tener incluida la gramática elaborada en la asignación de la tarea 1 en los archivos .jflex y .cup para generar los archivos creados por el analizador léxico y sintáctico.



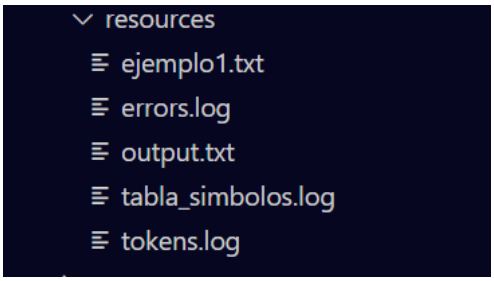
3. Ya con esto se puede pasar al momento de ejecución, ejecutando el programa en el archivo main que es el App.java



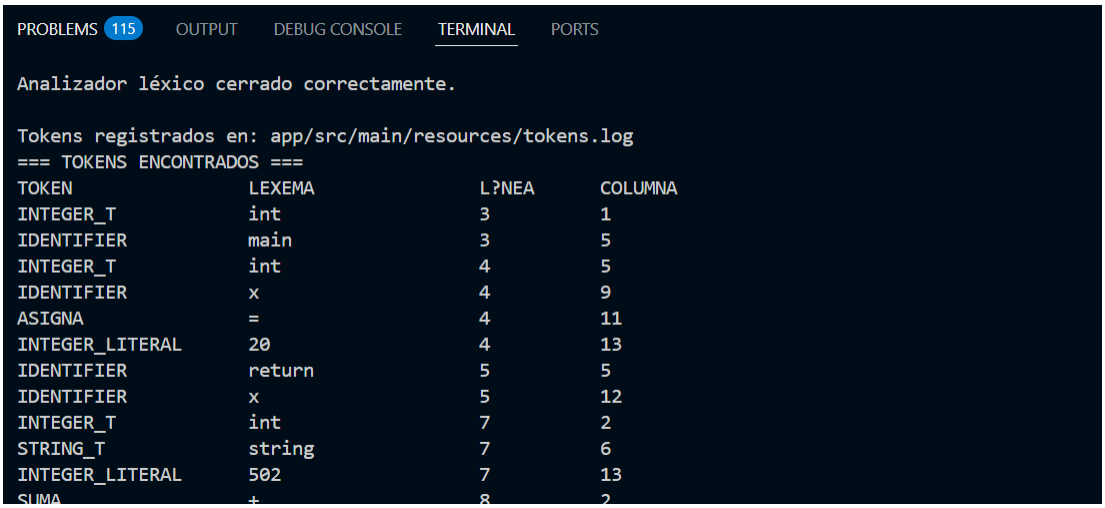
4 Inmediatamente el programa generara los archivos BasicLexerCup.java, parser.java y sym.java quienes son archivos generados por el Jflex y el Cup.



5. El programa en su tiempo de ejecución va generar los archivos de error, tabla de símbolos y los tokens.



6. El programa también mostrara por consola alguna información de los archivos generados.



Descripción del problema.

La etapa 3 del proyecto consiste en la generación de código destino, debe utilizarse el código 3D generado por el CUP durante el análisis semántico para, basado en el, generar código MIPS equivalente.

El problema encontrado en esta etapa es la cantidad de inconsistencias e incompletitudes que tiene el código intermedio generado, debe realizarse un trabajo importante en la re-generación de este código intermedio de manera que el código de tres direcciones resultante sea mucho mejor estructurado, considere aspectos como el tipo de dato que se maneja en cada caso
Y manejar bien los saltos y retornos s las etiquetas desde las cuales se realiza un salto.

Esta reestructuración permitiría que el código intermedio sea más limpio y congruente y además facilitaría el proceso de generación de código destino, lamentablemente no fue posible generar un código de tres direcciones sin afectar el funcionamiento del programa para las etapas 1 y 2 del proyecto.

Diseño del programa: decisiones de diseño, algoritmos usados.

Para esta etapa era necesario realizar una reestructuración completa de la generación de código intermedio, ya que el código intermedio obtenido durante el desarrollo del proyecto 2 no estaba estructurado adecuadamente para las necesidades de la generación de código MIPS.
El código intermedio que se genera actualmente utiliza el mismo tipo de variable "t#" independientemente del tipo de dato que se está manejando, sin embargo esto no es óptimo para la generación de código MIPS, es necesario implementar mecanismos de diferenciación entre variables que conformen un argumento de función para usar una nomenclatura más cercana al código destino, en lugar de "t#" usar algo como "v#", igualmente para cada tipo de dato que se cree debe usarse una etiqueta más acorde a lo que se necesita para la generación de código destino tomando en cuenta los tipos de registros que existen en MIPS.

Registros de Propósito General (32 registros)

(zero): Siempre contiene el valor 0, es de solo lectura
(at): Reservado para el ensamblador (assembler temporary)
(v0-$v1): Registros de valor de retorno de funciones
(a0-$a3): Registros de argumentos para llamadas a funciones
(t0-$t7): Registros temporales (no preservados en llamadas)
(s0-$s7): Registros guardados (preservados en llamadas a funciones)
(t8-$t9): Registros temporales adicionales
(k0-$k1): Reservados para el kernel del sistema operativo
(28(gp): Puntero global (global pointer)
(29(sp): Puntero de pila (stack pointer)
(fp o $s8): Puntero de marco (frame pointer) o registro guardado
(ra): Dirección de retorno (return address)

Registros Especiales

PC (Program Counter): Contiene la dirección de la siguiente instrucción a ejecutar
HI y LO: Almacenan los resultados de operaciones de multiplicación y división

HI: Parte alta del resultado
LO: Parte baja del resultado

Registros de Punto Flotante (32 registros)

(f0−f31): Para operaciones de punto flotante
Pueden usarse como registros de 32 bits (precisión simple) o combinarse en pares para 64 bits (doble precisión)

Función de cada tipo:

Temporales ($t): Para cálculos intermedios, no se preservan entre llamadas
Guardados ($s): Deben preservarse entre llamadas a funciones
Argumentos ($a): Pasan parámetros a funciones
Valores de retorno ($v): Devuelven resultados de funciones
Punteros especiales: Gestionan memoria y control de flujo

Librerías usadas: creación de archivos, etc.

Para la correcta elaboración del proyecto se utilizaron las siguientes librerías y herramientas:

- **Java Development Kit (JDK)**
- **JFlex** (generador de analizadores léxicos para Java).
- **CUP** (Constructor of Useful Parsers, generador de analizadores sintácticos).
- Java CUP Runtime (para ejecutar el parser generado por CUP)

Análisis de resultados: objetivos alcanzados, objetivos no alcanzados.

Objetivo	**Alcanzado**	**No Alcanzado**
El sistema debe leer un archivo fuente.	Si	
Se debe escribir en un archivo todos los tokens encontrados, identificador asociado con el lexema.	Si	
Por cada token deberán indicar en cuál tabla de símbolos va y cual información se almacenará.	Si	
Indicar si el archivo fuente puede o no ser generado por la gramática.		No alcanzado
Reportar y manejar los errores léxicos y sintácticos encontrados.	Maso menos (no se muestra cómo se debería de ver).	
Reestructuración de la generación del código intermedio		No logrado, rollback a la versión del proyecto 2, realizar este cambio implica cambios mayores a toda la funcionalidad existente del proyecto, principalmente desde la etapa 2.
Generación de código destino		No alcanzado, el analizador no genera el código, debe reestructurarse el análisis semántico y generación de código intermedio

Bitácora (autogenerada en git)

https://github.com/EscalanteWizard/Compi3.git

Puede acceder a una versión de documentación generada usando agentes de IA para ver aspectos específicos de este proyecto:

https://deepwiki.com/EscalanteWizard/Compi