Instituto Tecnológico de Costa rica

Escuela de Ingeniería en Computación

Compiladores e Intérpretes [IC - 5701]

Proyecto 1

Lexer y parser

Estudiantes:

Eduardo Rojas Gómez 2023152827

Yosimar Montenegro Montenegro 2023147365

Profesor(a) a cargo:

Allan Rodríguez Dávila

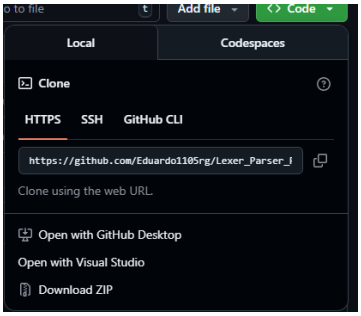
II Semestre | 2025

Enlace al repositorio del proyecto:

Eduardo1105rg/Lexer_Parser_PY1_Compi: Creación de un lexer y un parser mediante las herramientas Jflex y Cup, esto en JAVA.

Instrucciones de compilación y ejecución:

Lo primero paso se deberá de clonar el repositorio del proyecto o descargar el zip de este.



Una vez se tiene el proyecto en la computadora, se tiene que abrir desde un editor de código, una vez que se abre el proyecto se ejecutara los siguientes comandos en la terminar para compilar y ejecutar:

- Primero, ejecutar el comando *'./grandlew clean'* para limpiar el el .build, esto con el fin de poder crear los nuevos archivos.

- Después ejecutar el comando *'./grandlew build'* para generar el lexer y el parser.
- En caso de ser necesario en la capeta *'src/test/resources/test-programs'* se definen los archivos que se desean probar, es decir los archivo con la gramática del lenguaje que se está desarrollando.
- Ahora usando el comando *'./gradlew run --args="src/test/resources/test-programs/ejemplo1.txt"'* se ejecuta todo el programa.

Una vez ejecutado se producirá un archivo Tokens.txt en el cual se tienen los tokens y lexemas que se encontraron al leer el archivo de entrada, además en la consola se mostraran los errores léxicos y sintácticos que se encontraron al leer el archivo fuente, además después de los errores se imprimirán las tablas de símbolos que se formaron.

Descripción de problema:

Se solicita el desarrollo de un analizador léxico y un analizador sintáctico para la gramática del curso. Estos analizadores deben de ser desarrollados en Java utilizando las herramientas jflex y cup, en donde el jflex funcionara como lexer y el cup funcionara como el parser. La gramática del curso define las normas gramaticales para un lenguaje de paradigma funcional y tipado fuerte y estático.

Diseño del programa: decisiones de diseño, algoritmos usados:

Se desarrollo este proyecto utilizando grandle, en el archivo build.grandle se configuraron las task para la ejecución del parser y del lexer. Primero se decidió definir el lexer con todos los tipos terminales que se definió en la gramática diseñada para la tarea 1 del curso.

Después el archivo del parser se definió completamente en base al lexer y las producciones definidas. Primero se hizo una copia casi textual de la gramática original y posteriormente se fue adaptando poco a poco para que funcionara completamente en una definición que el cup pudiera entender.

La gramática definida es utiliza derivación por la izquierda y por derivación por la derecha, a la hora de realizar derivaciones y definir precedencia o expresiones que puede volver a tener múltiples expresiones mas veces, por ejemplo, los parámetros de una función, puede ser un solo parámetro o múltiples parámetros, y para permitir que sea uno o mucho se utiliza la recursividad para que sea posible que la gramática lo acepte.

Librerías usadas: creación de archivos, etc:

- Java.io: Librería para la entrada y salida.
- Jflex: Para la generación del analizador léxico.
- Cup: Para la generación del analizador sintáctico.
- Librias para tipos de dato de java: Adicionalmente se usan librería como List o ArrayList para trabajar con tipos de datos en java.

Análisis de resultados: objetivos alcanzados, objetivos no alcanzados, y razones por las cuales no se alcanzaron los objetivos (en caso de haberlos):

- **Objetivos alcanzados:**
    - Leer 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: Si.
    - Reportar y manejar los errores léxicos y sintácticos encontrados. Debe utilizar la técnica de Recuperación en Modo Pánico (error en línea y continúa con la siguiente). En el reporte de errores es fundamental indicar la línea en que ocurre: Si.
    - Reconocer la gramática: Si.
    - Generar el lexer: Si.
    - Generar el parser: Si.

Justificación de toma de decisiones: por qué se generaron las producciones en expresiones y la precedencia:

Para las expresiones, se opto por diseñar una estructura jerárquica que permita la representación de la semántica de operaciones aritméticas, lógicas y relacionales. La parte de expresiones aritméticas, también se dividió en subexpresiones (como expresioSuma, expresionProducto, expresionPotencia, expresionNumerica) con una especie de jerarquía, esto con el fin de tener una estructura recursiva y con prioridad de operaciones.

También se agregaron producciones para lo que serian expresiones unarias y agrupaciones con paréntesis, además, este último también se maneja de forma que permita representar una jerarquía de paréntesis, es decir, que un paréntesis pueda tener más paréntesis adentro y que un paréntesis se resuelva antes que otras operaciones.

Ahora, con respecto al orden de precedencia, se define que la operación de potencia tiene mayor precedencia que los operadores de multiplicación, división y modulo, y estos tienen mayor precedencia que las sumas y las restas.

Por otro lado, se define que las operaciones relacionales se deben de evaluar después de las operaciones aritméticas, y los operadores lógicos se evalúan al final en donde NOT es el más fuerte. Por último, la asignación tiene la precedencia más baja para hacer que las expresiones del lado derecho se realicen antes de asignarse.

Adicionalmente, se utilizan estructuras recursivas para los casos en que una producción (como la de parámetros de una variable) ocupe realizarse múltiples veces.