Instituto Tecnológico de Costa rica

Escuela de Ingeniería en Computación

Compiladores e Intérpretes [IC - 5701]

Proyecto 2

Análisis semántico y código intermedio

Estudiantes:

Eduardo Rojas Gómez 2023152827

Yosimar Montenegro Montenegro 2023147365

Profesor(a) a cargo:

Allan Rodríguez Dávila

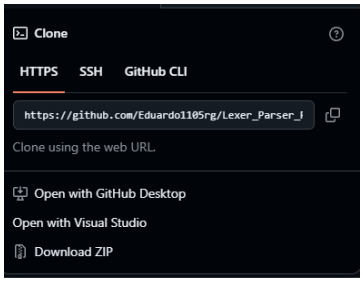
II Semestre | 2025

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:

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



Como se muestra en la imagen adjunta, en esa parte de a página de git se realiza este proceso.

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 .build, esto con el fin de poder crear los nuevos archivos.

- Después ejecutar el comando ‘./grandlew generateLexer’ para generar el lexer y posteriormente el comando ‘./grandlew generateParser’ para generar 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, sintácticos y semánticos que se encontraron al leer el archivo fuente, además después de los errores se imprimirán las tablas de símbolos que para el programa.

Adicionalmente se genera un archivo con el código intermedio en formato tres direcciones basado en el archivo fuente.

Descripción del problema:

Se solicita el desarrollo de un analizador semántico (para un lenguaje de tipado fuente y explicito) y la generación del código intermedio en tres direcciones, lo anterior implementando y corrigiendo el proyecto 1. Esto mediante las herramientas jflex y cup.

Diseño del programa: decisiones de diseño y algoritmos usados:

Este proyecto se desarrolló utilizando grandle y en el archivo build.grandle se configuraron las tareas para la ejecución del proyecto.

Basado en lo que ya se tenia del proyecto 1, se procedió con el desarrollo del proyecto 2. En este proyecto se reconstruyeron las producciones de expresiones para solucionar muchos problemas que se presentaban en esta sección, en si se pasaron las partes de char, string y booleanos a la parte de expresiones numéricas, con esto se logro que las expresiones tuvieran un solo camino y no dieran tanto shift/reduce y tuviéramos problemas a la hora de derivar.

Para la construcción de una nueva tabla de símbolos, se decidió eliminar la anterior y se empezó con una nueva, ahora, basado en un PDF compartido por el profe se creó una clase ámbitos y clases de tokens. El ámbito es lo que se crea al entrar a algo que tenga un bloque o su propio espacio, como los seria una función, una estructura de control o algo así, además, en estos ámbitos se tiene una lista la cual va a almacenado los tokens los cuales son las variables que va registrando el sistema. En esos tokens se almacena cosas como el nombre del identificador o el tipo de datos. Además de esto se crean otros tipos de tokens los cuales son usados para mover valores en los RESULT de las producciones de una forma más sencilla.

Para la parte del análisis semántico en la mayoría de las producciones lo que se hace es obtener el tipo de las expresiones, que viene en una clase de token especial para esta validación y lo comparamos con el esperado en donde lo ocupemos.

Librerías usadas: creación de archivo, lexer y parser, etc:

- Java.io: Librería para la entrada y salida.

- Jflex: Para la generación del analizador léxico.

- Cup: Para la generación del analizador sintáctico.

- Librerías 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:**
    - Gramática, scanner y parser: Si.
    - Analizador semántico: Si.
    - Generación de código intermedio: Si.

Justificación de toma de decisiones:

Para el cambio realizado en expresión se debió mas que nada a que en muchas partes se tenía que llegar hasta lo que es el identificador, por lo tanto, si manejaba aritmética, string char y boolean por separados estos daban, problemas, por lo tanto, se unificaron en una sola producción, lo que corregía el error, lo único es que se tenían que hacer más validaciones semánticas para que esto fuera posible.

Para la generación de la tabla de símbolos, se decidió registrar em una clase ámbitos por recomendación de un compañero, el cual nos comentos que era mejora manejar esto por clase, lo cual hacía que fuera más sencillo manejarlos, por eso basado en un PDF compartido por el profesor se creo que clase para el ámbito. Adema los tokens que representan a las variables o las funciones, se creó una clase token la cual hacía que fuera más fácil registrar y manejar información.

Para lo que son las validaciones semánticas, estas se realizan en cada una de las producciones en donde se ocupa acceder a un id o validar tipos.

En cuanto al código intermedio, básicamente buscábamos convertir la correctitud semántica del CUP en cuádruplas (Cuads en código) que representan todas las instrucciones como asignaciones, control de flujo, llamadas, listas, etc. Básicamente tenemos estructuras Cuads y un TAC_Generator que tiene contadores para etiquetas y temporales. Tenemos listas globales y generadores de TAC para cada instrucción. Buffers para los bloques, con los que se simula algo parecido a los autómatas de pila, lo usamos para reconocer si estamos en un bloque o no.