Seminario de Lenguajes Formales
Facultad de ingeniería
Universidad de Palermo

Hugo Centurión (85594) Profesora: Belén Alegre

TP Integrador: Compilador de LenguajeX

Descripción del Proyecto

El proyecto consiste en el desarrollo de un compilador para un lenguaje de programación ficticio denominado “LenguajeX” utilizando el programa ANTLR y java para su implementación.

Este compilador se encarga de realizar las etapas de compilación de análisis sintáctico con generación del árbol de derivación, análisis semántico y generación del código intermedio en formato JavaScript.

Acciones preliminares (ANTLR)

Se utilizó ANTLR para generar las clases necesarias que se incorporarán en nuestro proyecto java. ANTLR es una poderosa herramienta para reconocer y analizar de lenguajes utilizando algoritmos de parsing descendiente (Left to right scan and leftmost derivation o LL) a partir de una gramática dada.

1. **Generación de clases:**
    - Se creó una gramática basada en el código de ejemplo proporcionado por la consigna del TP y se guardó en un archivo de texto llamado: *‘Lenguajex.g4’*.
    - Se instaló el programa ANTLR y se generaron las clases que luego se incorporarán al proyecto de compilador en java.

Proyecto Compilador en Java (desarrollado en IntelliJ IDEA)

A continuación se describe la lógica principal del compilador, resumida en las siguientes etapas presentes en el archivo ‘*main.java’*:

1. **Análisis Léxico:**
    - Se crea un lexer que convierte el código fuente en una secuencia de tokens.
    - Se reemplaza el manejador de errores por defecto con uno personalizado para capturar y reportar errores de sintaxis de tipo lexico-gráfico para enviar un mensaje de error personalizado.

2. **Análisis Sintáctico:**
    - Se crea un parser que toma los tokens y genera un árbol sintáctico (parse tree).
    - Se reemplaza el manejador de errores por defecto con uno personalizado para capturar y reportar errores de sintaxis para enviar un mensaje de error personalizado.

3. **Generación del Árbol Sintáctico:**
    - Si no hay errores de sintaxis, se genera una representación gráfica del árbol sintáctico y se guarda como una imagen con el nombre: '*arbol_sintactico.png' en el directorio raíz del proyecto*.

4. **Análisis Semántico:**
    - Se crea un listener personalizado: '*LenguajexCustomListener.java'* a partir de la clase base que genera ANTLR para extender su funcionalidad sin modificar la misma. Éste monitorea los eventos mientras se recorre el árbol sintáctico.
    - Se capturan y reportan los errores semánticos encontrados. (A modo de ejemplo se implementa sólo validación de variables no declaradas pero se pueden agregar más validaciones dentro del listener personalizado utilizando la misma mecánica).

5. **Generación de Código Intermedio:**
    - En caso de no encontrarse errores semánticos (o si se decide continuar a pesar de ellos), se genera el código intermedio. Éste código se produce a medida que se recorre el árbol sintáctico por lo que la lógica para esto se implementa también en el archivo '*LenguajexCustomListener.java',* si se desea se puede realizar en una nueva clase extendiendo la clase base de la misma forma y así separar ambos procesos y llevar una codificación mas prolija y modularizada. (Nota: Se produce el código en formato de JavaScript sólo a modo de ejemplo para el trabajo, pero lo más común es hacerlo en otros lenguajes de mas bajo nivel como por ejemplo: Código de tres direcciones o Assembler).
    - El código generado se guarda en un archivo llamado '*codigo_intermedio.js' en el directorio raíz del proyecto*.

6. **Finalización:**
    - Se informa al usuario sobre el éxito o fracaso del proceso de compilación.

Conclusiones

- **Eficacia del Proceso de Compilación:** El compilador es capaz de detectar y reportar tanto errores sintácticos como semánticos lo cual es crucial para asegurar la corrección del código antes de la generación del código intermedio.

- **Extensibilidad y Modularidad:** La estructura del compilador en este proyecto permite una fácil extensión para soportar nuevas características del lenguaje o mejorar las existentes. Separar las fases de análisis sintáctico, semántico y la generación de código en diferentes componentes (lexer, parser, listener) contribuye a la claridad y mantenibilidad del código.

- **Utilidad Práctica:** Aunque el “LenguajeX” es ficticio, el proyecto ayuda a ejercitar y demostrar conceptos clave en el desarrollo de compiladores que pueden aplicarse a lenguajes de programación reales, proporcionando una base sólida para proyectos más complejos en el futuro.