Instituto Tecnológico de Costa Rica

Unidad de Computación

Compiladores E Intérpretes

Verano I 2023

Primer Proyecto

Profesor

Allan Rodriguez Davila

Estudiantes

Deivid Matute Guerrero

Joshua Sancho Burgos

Centro Académico de Limón

07 de Diciembre del 2023

Manual de Usuario

Creación del Lexer:

Para programar el lexer es necesario instalar las librerías JFlex y CUP. Una vez instaladas e implementadas en el proyecto se procede a crear un archivo en formato ".jflex" ,".flex", o ".lex" que son los formatos compatibles con la librería JFlex. Este archivo se divide en 3 secciones, el código de usuario (donde se realizan configuraciones, se declara el paquete y se realizan los imports), opciones y declaraciones (donde se declaran los macros y estados) y las reglas léxicas (donde se especifican los tokens que regresará el lexer). Es importante que la configuración "%cup" esté incluida para ser compatible con el parser.

Creación del Parser:

Para programar el parser es necesario instalar e implementar la librería CUP. Una vez terminada la programación del lexer se procede a programar el parser. Primero se realizan los ajustes necesarios para la compatibilidad con JFlex (imports y scanner), luego, para esta primera parte, se declaran los símbolos no terminales. Estos deben ser los mismos que se declararon en el lexer, sino, se generará un error, al no existir tal símbolo en dado caso. Para generar el parser es necesario que exista un símbolo no terminal que de entrada al programa.

Generación de Parser y Lexer:

Para generar el lexer y el parser se tienen 2 opciones.

La primera es ejecutar la función "GenerarLexerParser()" en el archivo "App.java'. Esta función elimina los archivos "lexer.java, "parser.java" y "sym.java" si son archivos existentes en el sistema, en caso contrario, los genera si se encuentran los archivos "lexer.flex" y "parser.cup" en las carpetas correspondientes y los mueve de su localización hacia la carpeta "ParserLexer"; en caso de que estos archivos no existan (en la última actualización del proyecto, estos archivos sí existen), se deberá localizar el archivo MainJFlexCup.java y comentar la línea 4, que contiene lo siguiente: "import ParserLexer.lexer;", además de comentar la función "LexerTest" completa, ya que estas partes del código requieren de la existencia del archivo "lexer.java".

La otra forma es generarlos manualmente por medio de los comandos del manejador de paquetes Maven, con "mvn jflex:generate" para generar a "lexer.java" y "mvn

cup:generate” para generar a “parser.java” y “sym.java”, sin embargo, no se recomienda usar esta opción, ya que localiza los archivos en target y no los mueve a la carpeta “ParserLexer”, por lo que se pueden ocasionar algunas incompatibilidades.

Pruebas de funcionalidad

Mediante el uso de la librería JUnit5, que incluye **“**org.junit.jupiter.api.Assertions” y “org.junit.jupiter.api.Test” se probó que cada uno de los tokens fuera reconocido por el lexer producido en JFlex y que coincidiera con el Symbol producido por CUP.

```java
package org.compiler;

import java_cup.runtime.Symbol;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;

import static org.junit.jupiter.api.Assertions.assertThrows;

Joshua Sancho
public class LexerTest {

    Joshua Sancho
    @Test
    public void matchPINO() throws IOException {
        String testString = ",";
        lexer lex = new lexer(new StringReader(testString));
        Symbol symbol = lex.next_token();
        Assertions.assertEquals(sym.PINO, symbol.sym);
```

Descripción del problema

Contexto:

En el desarrollo de compiladores, el análisis léxico es una fase crucial que consiste en convertir el flujo de caracteres del código fuente en una secuencia de tokens, que son unidades léxicas con significado en el lenguaje de programación. En este escenario, se utiliza JFlex para la generación del lexer (analizador léxico) y CUP para el análisis sintáctico.

Problema:

El problema radica en implementar un lexer eficiente utilizando JFlex que sea capaz de reconocer y clasificar correctamente los tokens del lenguaje de programación objetivo. Estos tokens pueden incluir palabras clave, identificadores, literales, operadores, delimitadores, comentarios, entre otros.

Desafíos Específicos:

- Manejo de Palabras Clave: Identificar y clasificar correctamente las palabras clave del lenguaje. Esto implica definir patrones que distingan entre identificadores y palabras clave de manera precisa.
- Reconocimiento de Literales: Implementar reglas para el reconocimiento de literales, como cadenas de texto, números enteros y decimales, caracteres especiales, etc.
- Gestión de Comentarios: Desarrollar patrones para identificar y manejar comentarios de una o varias líneas, así como comentarios de una sola línea.
- Manejo de Espacios en Blanco: Ignorar espacios en blanco y saltos de línea cuando no afecten la estructura del código. No obstante, estos deben ser tenidos en cuenta en situaciones donde su omisión podría causar ambigüedades.
- Manejo de Errores: Implementar mecanismos para reportar errores léxicos de manera clara, indicando la posición en la que se detectó el error y describiendo la naturaleza del mismo.

Objetivo Final:

La solución al problema debe proporcionar un lexer robusto y eficiente que sea capaz de analizar el código fuente del lenguaje objetivo, generando una secuencia de tokens que puedan ser consumidos por el analizador sintáctico desarrollado con CUP. Este

conjunto de herramientas permitirá la construcción de un compilador capaz de procesar y comprender el código fuente del lenguaje de programación específico.

Diseño del programa

Especificaciones Iniciales:

- Identificación de las palabras clave, operadores, delimitadores y otros elementos léxicos del lenguaje de programación objetivo.
- Definición de patrones mediante expresiones regulares en JFlex para reconocer cada tipo de token.

Implementación del Lexer con JFlex:

- Utilización del algoritmo de análisis léxico de JFlex, basado en autómatas finitos deterministas (AFD).
- Creación de un archivo ".jflex" con las reglas de análisis léxico.
- Desarrollo de patrones para reconocer y clasificar palabras clave, identificadores, literales, operadores, delimitadores y comentarios.
- Manejo de espacios en blanco y saltos de línea según las necesidades del lenguaje.

Generación del Analizador Sintáctico con CUP:

- Implementación del algoritmo LALR (Look-Ahead Left-to-Right) utilizado por CUP para el análisis sintáctico.
- Definición de las reglas gramaticales en un archivo ".cup".
- Especificación de la estructura del árbol sintáctico y las acciones asociadas a cada regla.
- Integración del lexer de JFlex con el analizador sintáctico de CUP para la construcción del árbol.

Generación de función LexerTest:

- Recepción como argumento de la ruta del archivo que contiene el código fuente a analizar.
- Verificación de que el lexer sea capaz de reconocer todos los tokens del lenguaje y clasificarlos correctamente.
- Generación de mensajes de error descriptivos cuando se produce un error léxico.
- Lectura del archivo para generar una secuencia de tokens para mostrarlos en consola.

Manejo de Errores:

- Generación de mensajes de error claros y descriptivos utilizando técnicas de análisis de contexto.

Pruebas y Depuración:

- Creación de conjuntos de pruebas exhaustivos que cubran diversos aspectos del lenguaje.
- Ejecución de pruebas para verificar la correcta identificación y clasificación de tokens.
- Depuración de errores y ajuste de las reglas léxicas y gramaticales según sea necesario.

Documentación:

- Creación de documentación clara y detallada que describa el funcionamiento del lexer y el analizador sintáctico.
- Inclusión de ejemplos de uso y casos de prueba en la documentación.

Integración en un Compilador:

- Integración del lexer y del analizador sintáctico en un compilador completo capaz de transformar el código fuente en código ejecutable.

Librerías usadas

- JFlex: Es un generador de analizadores léxicos (conocidos también como escáner) para Java, escrito en Java. Este toma como entrada un conjunto de expresiones regulares y acciones correspondientes. Estos se basan en autómatas finitos deterministas (DFAs), los cuales son muy veloces y no requieren de un backtracking pesado. Está diseñado para trabajar con el generador de parsers LALR CUP.
- CUP: Es un generador de parsers LALR. En este se pueden especificar los símbolos de una gramática propia, así como producciones y acciones. Este funciona especialmente bien con JFlex.
- JUnit5: Esta librería funciona como una base para lanzar frameworks de pruebas en la JVM. También incluye una API de motor de pruebas.

Análisis de resultados

Tarea / Requerimiento	*Estado*	*Observaciones*
Operadores aritméticos binarios	Completado	
Operadores aritméticos unarios	Completado	
Operadores relacionales	Completado	
Operadores lógicos	Completado	
Identificador	Completado	
Tipos	Completado	
Literales	Completado	
Paréntesis	Completado	
Paréntesis cuadrado	Completado	
Llaves	Completado	
Lexemas de estructuras de control	Completado	
Print, read	Completado	
Lexema de fin de expresión	Completado	
Lexema de asignar	Completado	
Lexema separador	Completado	
Generación y localización de lexer y symbol	Completado	
Test de lexer	Completado	

Bitácora

Repositorio de Github:

https://github.com/JoshSan14/Xmas-Compiler