|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| Manual Técnico  Organización de Compiladores  Universidad de San Carlos de Guatemala |
| |  |  |  | | --- | --- | --- | | Javier Monterroso | 201700831 | Facultad de Ingeniería | |

Contents

[Análisis léxico usando JFlex 2](#_Toc1676813)

[Generador: 2](#_Toc1676814)

[Análisis Sintáctico con Cup 3](#_Toc1676815)

[Generador: 3](#_Toc1676816)

# Análisis léxico usando Flex

La primera fase de la compilación es el análisis léxico: la descomposición de la entrada en tokens.

Un token generalmente se describe mediante un número entero que representa el tipo de token, posiblemente junto con un

atributo, que representa el valor del token. Por ejemplo, en la mayoría de los lenguajes de programación nosotros

Cada palabra reservada o símbolo especial se considera un tipo diferente de token, en la medida en que

Se distinguen por un entero diferente para representar su tipo.

Diferentes identificadores tienen el mismo tipo, y se distinguen por tener un diferente

atributo (tal vez el texto que compone el identificador, o un índice entero en una tabla de identificadores).

Se utilizó esta librería para generar el Scanner o analizador léxico del proyecto utilizando el siguiente código para generar la clase

## Generador:

public static void main(String[] args)

{

String path="src/Analizadores/a\_Lexico.jflex";

generarLexer(path);

}

public static void generarLexer(String path)

{

File file=new File(path);

jflex.Main.generate(file);

}

Se utilizó el generador por defecto de la librería y se compiló en java

# Análisis Sintáctico con Bison

El uso de CUP implica la creación de una especificación simple basada en la gramática para la cual se necesita un analizador, junto con la construcción de un escáner capaz de dividir los caracteres en símbolos significativos (como palabras clave, números y símbolos especiales).

La especificación contiene cuatro partes principales.

La primera parte proporciona declaraciones preliminares y misceláneas para especificar cómo se generará el analizador y proporcionar partes del código de tiempo de ejecución. En este caso, indicamos que las clases java\_cup.runtime deben importarse, luego suministramos un pequeño bit de código de inicialización y algo de código para invocar al escáner para recuperar el siguiente token de entrada. La segunda parte de la especificación declara terminales y no terminales, y asocia las clases de objetos con cada una. En este caso, los terminales se declaran como sin tipo o de tipo Integer. El tipo especificado de terminal o no terminal es el tipo del valor de esos terminales o no terminales. Si no se especifica ningún tipo, el terminal o no terminal no tienen ningún valor. Aquí, ningún tipo indica que estos terminales y no terminales no tienen ningún valor.

La tercera parte especifica la precedencia y la asociatividad de los terminales.

La última declaración de precedencia otorga a sus terminales la más alta precedencia. La parte final de la especificación contiene la gramática.

Se utilizó esta librería para generar el Parser o analizador sinractico del proyecto utilizando el siguiente código para generar la clase

## Generador:

String opciones[] = new String[7];

//Seleccionamos la opción de dirección de destino

opciones[0] = "-destdir";

//Le damos la dirección, carpeta donde se va a generar el parser.java & el simbolosxxx.java

opciones[1] = "src/Analizadores";

//Seleccionamos la opción de nombre de archivo simbolos

opciones[2] = "-symbols";

//Le damos el nombre que queremos que tenga

opciones[3] = "sym";

//Seleccionamos la opcion de clase parser

opciones[4] = "-parser";

//Le damos nombre a esa clase del paso anterior

opciones[5] = "parser";

//Le decimos donde se encuentra el archivo .cup

opciones[6] = "src/Analizadores/a\_Sintactico.cup";

try

{

java\_cup.Main.main(opciones);

}

catch (Exception ex)

{

System.out.print(ex);

}