Contenido

[Implementación de un Scanner y Parser Usando Java Cup y Jlex 3](#_Toc396962206)

[Planteamiento del problema 4](#_Toc396962207)

[Importancia de los compiladores/intérpretes 5](#_Toc396962208)

[Solución 6](#_Toc396962209)

[Analizador Léxico 7](#_Toc396962210)

[Estructura 7](#_Toc396962211)

[Clase Token 7](#_Toc396962212)

[Macros 7](#_Toc396962213)

[Directivas JLex 8](#_Toc396962214)

[Buffer 8](#_Toc396962215)

[Expresiones regulares (reglas) 8](#_Toc396962216)

[Manejo de errores 9](#_Toc396962217)

[Clase Main 9](#_Toc396962218)

[Generación del scanner 9](#_Toc396962219)

[Ejecución del scanner 10](#_Toc396962220)

[Salida del programa 10](#_Toc396962221)

[Analizador Sintáctico 11](#_Toc396962222)

[Actualizaciones al scanner 11](#_Toc396962223)

[Estructura 11](#_Toc396962224)

[Importar librería cup 11](#_Toc396962225)

[Action code 11](#_Toc396962226)

[Parser code 11](#_Toc396962227)

[Declaración de Símbolos terminales 12](#_Toc396962228)

[Declaración de Símbolos no terminales 12](#_Toc396962229)

[Precedencia de operadores 12](#_Toc396962230)

[Producciones 12](#_Toc396962231)

[Clase Main 14](#_Toc396962232)

[Generación del parser 15](#_Toc396962233)

[Ejecución del scanner 15](#_Toc396962234)

[Salida del programa 16](#_Toc396962235)

[Video de la ejecución del parser y scanner. 16](#_Toc396962236)

[Conclusiones 17](#_Toc396962237)

[Referencias 17](#_Toc396962238)

# Implementación de un Scanner y Parser Usando Java Cup y Jlex

Cristopherson Tadeo Torres Martínez

Materia Compiladores, Maestría en Ciencias Computacionales, Universidad Autónoma de Guadalajara, Zapopan, Jalisco, México.

E-mail: cristopherson.torres@gmail.com

Enviado: 28 de Agosto del 2014

# Planteamiento del problema

Hoy en día muchas de nuestras actividades cotidianas son ejecutadas por dispositivos electrónicos como computadoras, tablets, dispositivos móviles, refrigeradores con microprocesadores que permiten adquirir la información de los hábitos del usuario para que se le provean un conjunto de opciones de acuerdo a su perfil y preferencias. Sin embargo todos estos dispositivos necesitan ser programados para que puedan tener una secuencia de instrucciones bien definidas y se puedan aprovechar por parte del usuario común sin necesidad de conocer el funcionamiento básico de dichos dispositivos. De igual manera los ingenieros que se encargan de programar dichos dispositivos necesitan usar un conjunto de instrucciones propio de la máquina que está usando. Estas instrucciones llegan a ser complejas Es aquí donde se ve la necesidad de usar otras herramientas para que se agilice el proceso de programación de dispositivos electrónicos el cual al usarse instrucciones complejas impacta directamente en los tiempos de producción de dichos productos. Sin embargo dichas herramientas pueden ser muy costosas o no se logra encontrar una que esté a la medida de las necesidades del usuario por lo que se necesita la creación de una herramienta por parte de los interesados para agilizar a largo plazo los tiempos de producción de código. En este caso de uso se explora el uso de un lenguaje ya definido llamado Cool (Classroom Object Oriented Language) el cual nos servirá para la implementación de ambos un analizador léxico y un analizador sintáctico.

# Importancia de los compiladores/intérpretes

Un traductor es un programa que toma un programa escrito en un lenguaje, llamado fuente, y genera un programa equivalente en otro lenguaje, llamado objeto [1].

Un intérprete es un programa que acepta un programa escrito en lenguaje fuente y lo ejecuta. Ej. PROLOG, SQL, etc. Un intérprete NO produce código en lenguaje objeto, solo ejecuta el programa fuente mismo [1].

La importancia de los compiladores radica en que mejora la productividad del software al ocultar detalles de bajo nivel [2]. La generación de nuevos lenguajes radica en cubrir huecos que otros lenguajes que fueron usados primeramente dejaron descubiertos. Cuando un lenguaje es creado ataca las necesidades que han sido detectadas por parte de los encargados de construir dicho lenguaje. Sin embargo, otras entidades con necesidades diferentes pueden ver factores bloqueantes al usar el lenguaje con las limitaciones que se han detectado por lo tanto surge la necesidad de cubrir esas limitaciones con un lenguaje que se enfoque en solucionar esos problemas y con ello poco a poco van surgiendo nuevos lenguajes. De igual manera un intérprete ayuda a la ejecución de un código que no está diseñado para el ambiente en el cual está instalado. En estos aspectos tenemos caso de estudio a java que se ejecuta sobre una máquina virtual instalada en la máquina anfitriona, el intérprete de Java lee el código generado y entonces lo ejecuta en la máquina virtual. Esto trae como consecuencia que la aplicación es portable entre diferentes ambientes pero se sacrifica desempeño de ejecución [3]

# Solución

Dependiendo de las necesidades y el ambiente del usuario se pueden usar diferentes herramientas, la más común es lex [4] y yacc [5]. Sin embargo para el caso de uso propuesto se usaran las contrapartes vinculadas a Java: Jlex [6] y java\_cup [7]

Una vez provisto el manual de uso del lenguaje Cool se empieza con el análisis del alfabeto y conjunto de tokens permitidos por el lenguaje. Este paso inicial consiste en crear el analizador léxico (scanner).

## Analizador Léxico

Como se había mencionado antes, se usará jlex para generar un scanner basado en java. Todos los puntos mencionados en ésta sección se pueden localizar dentro del archivo cool-lex

### Estructura

Las especificaciones de Jlex indican que se organiza en tres secciones separadas por doble signo de porcentaje [6]

1. Código de usuario: el cual es usado para la definición de los tokens que son esperados en el lenguaje por medio del uso de una clase privada llamada Token.
2. Directivas de JLex: Los posibles estados dentro del scanner son definidos así como algunas macros de uso común.
3. Expresiones regulares (reglas): esta sección contiene las reglas de análisis léxico.

### Clase Token

Esta clase fue creada para asignarle una etiqueta a las expresiones regulares encontradas y así generar la tabla de símbolos. Esta clase es accesible en el archivo cool-lex del proyecto scanner.

### Macros

En la siguiente sección del archivo cool-lex se encuentran algunas macros definidas como la siguiente.

CASE = [Cc][Aa][Ss][Ee]

DARROW = =>

WHITESPACE = ({BACKSPACE}|{CARRIAGE}|{FORMFEED}|{NEWLINE}|{TAB})

NONNEWLINE\_WHITE\_SPACE\_CHAR=[\ \t\b\012]

Esto con el fin de brindar algunas definiciones preestablecidas antes de escribir las reglas de las expresiones regulares y que no sea tan complejo de leer o mantener.

### Directivas JLex

Algunas directivas de JLex son usadas para el mejor reporteo del estado del análisis léxico.

%line

Para que se pueda tener acceso al número de línea e identificar en qué punto del archivo se detectó el error léxico.

%type Token

Ayuda a cambiar el nombre del tipo de dato regresado por el scanner. En éste caso se usa la clase Token.

%state READCHAR

%state READCOMMENT

%state READONELINECOMMENT

Estos son diferentes etiquetas que permiten cambiar de estados dentro de la máquina de estados. Por ejemplo READCOMMENT y READONELINECOMMENTayudan a agrupar todo el contenido dentro de un comentario. READCHAR permite formar cadenas de texto.

%eofval

Define qué valor regresar cuando se llega al final de archivo. En este caso se usa null para indicar al programa principal que el análisis ha terminado.

### Buffer

Un buffer es definido para que cada token que el scanner está detectando sea almacenado en una variable temporal y permita formar cadenas de texto.

private String str\_buffer = "";

### Expresiones regulares (reglas)

En esta sección se definen las reglas a seguir para formar los Tokens que se van reconociendo.

<YYINITIAL> \+ { return new Token(yytext(), Token.OPERATOR); }

En la primer parte se indica que se está en el estado inicial. Esto es muy útil para poder determinar cuándo se está formando una cadena de texto o se están leyendo comentarios que debieran ser omitidos.

En la segunda parte se retorna el objeto Token que ha sido localizado.

### Manejo de errores

En caso de haber un símbolo inválido una impresión en consola reportará lo sucedido.

Esto se puede ver con la siguiente regla dentro del archivo cool-lex

[^] { System.err.println("Invalid sequence at line " + (yyline + 1) + " : "+ yytext()); return new Token(yytext(), Token.ERROR);}

Donde el símbolo ^ indica que todo lo que no queda en las expresiones regulares mencionadas previamente se deben de considerar como inválido.

### Clase Main

Esta clase es usada para ejecutar el scanner y recibir el archivo a analizar, esta clase se localiza en el archivo Main.java

class Main {

public static void main(String args[]) throws java.io.IOException {

Yylex lex = new Yylex( new FileInputStream(args[0]));

Token result = lex.yylex();

while ( result != null ) {

System.out.println("(" + result.getType() + ") " + result.getToken());

result = lex.yylex(); //get next token

} } }

Se toma el archivo de entrada, se invoca al método yylex para sacar el primer token. En caso de que sea válido se imprime y se itera hasta que ya no haya más elementos a escanear.

### Generación del scanner

Para generar el scanner usando JLex en Linux el siguiente comando es usado.

jlex cool-lex

Una vez generado se debe compilar el código resultante.

javac cool-lex.java

Por último se compila el código del archivo Main.java

javac Main.java

### Ejecución del scanner

Para que se ejecute el scanner solo se necesita ejecutar el siguiente comando

java Main $archivo

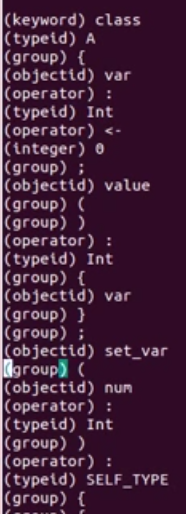
Donde $archivo es el nombre del archive a escanear. En el proyecto se provee un conjunto de archivos de prueba provistos por coursera.

Todos estos pasos son ejecutados automáticamente por el script autorun.sh.

./autorun.sh

### Salida del programa

La salida del programa es un archivo al que se le concatena la extensión .txt y el cual contiene la tabla de símbolos como se puede ver a continuación:



## Analizador Sintáctico

Para el analizador sintáctico se optado por usar java\_cup [7] ya que es compatible con jlex. Todos los puntos mencionados en ésta sección se pueden localizar dentro del archivo cool.cup

### Actualizaciones al scanner

Para una mejor adaptación con el código de java\_cup algunas modificaciones fueron hechas al scanner en comparación con el primer entregable:

1. El archivo del scanner ha sido renombrado de cool-lex a CoolScanner
2. La clase Token ha sido separada del archivo y radica en su propio archivo Token.java.
3. La directiva %cup ha sido agregada para compatibilidad con java\_cup
4. Funciones para devolver la línea del archivo que se está analizando y el nombre del archivo ha sido proporcionado con el fin de que se provea más información al parser cuando un error es detectado
5. Las macros READONELINECOMMENT y READCOMMENT son ignorados. A menos que el carácter EOF sea detectado antes de cerrar el comentario. En ese caso un error será reportado.
6. Ya no se genera un archivo de salida con la tabla de símbolos, simplemente el parser trata de revisar que la gramática sea correcta y en caso de fallo será reportado. Ya sea por error sintáctico o léxico.

### Estructura

La estructura de un archivo cup es más compleja que para el scanner. Se compone de los siguientes elementos:

### Importar librería cup

Es necesario especificar que se debe importar la librería de cup

import java\_cup.runtime.\*;

### Action code

Esta sección es usada para extraer información de importancia para el reporte de errores, tales como la línea en la que el error se detectó y el nombre del archivo que falló.

action code {: code :}

### Parser code

Esta sección es usada para reportar errores desde dentro de la clase que hace el análisis sintáctico.

parser code {: code :}

### Declaración de Símbolos terminales

Aquí los símbolos que son extraídos por el scanner (representados con la clase Token) son determinados como terminales y por lo tanto se espera que se encuentren como componentes del lenguaje.

terminal Token TYPEID, OBJECTID;

### Declaración de Símbolos no terminales

Aquí se le asigna un tipo de dato que le da valor semántico a las producciones que se van generando.

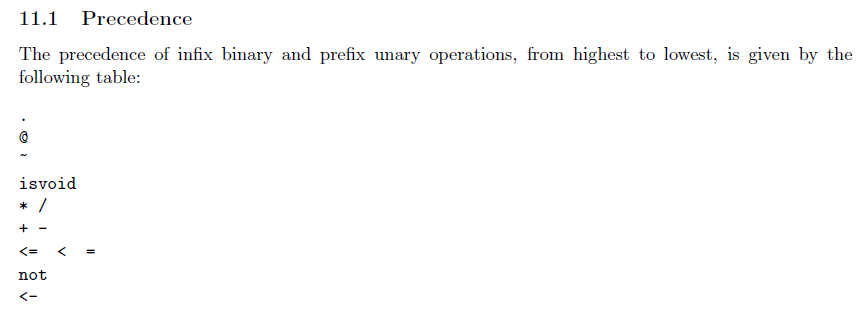
non terminal Program prog;

### Precedencia de operadores

Esta sección es obligatoria para evitar ambigüedad cuando diferentes operadores son hallados dentro de las producciones

precedence left DOT;

La precedencia es definida por el archivo cool\_manual



### Producciones

Finalmente vienen las producciones las cuales indican las reglas gramáticas a seguir para indicar que el código está siguiendo la estructura definida por el lenguaje.

prog ::= class\_expr:c

{:

Program p = new Program();

p.setExpression(c);

p.setTokenList(tokenList);

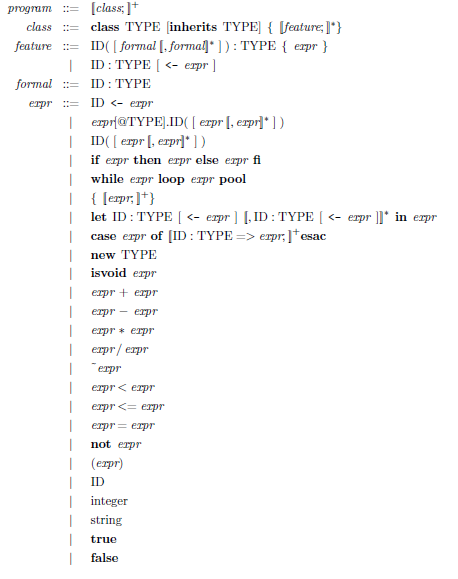
RESULT = p;

:}

;

Muchas más producciones son agregadas. Están disponible en el archivo cool.cup

Las producciones son definidas por el archivo cool\_manual.pdf. A continuación una figura que ejemplifica dichas reglas.



### Clase Main

Esta clase es la misma usada por el scanner en el primer entregable se localiza en el archivo Main.java

import java\_cup.runtime.Symbol;

class Main {

public static void main(String args[]) throws java.io.IOException {

CoolScanner scanner = new CoolScanner( new FileInputStream(args[0]));

scanner.setCurrentFileName(args[0]);

CoolParser parser = new CoolParser(scanner);

Program program = null;

try {

program = ((Program)parser.parse().value);

//program.printTree();

//program.printProgram();

System.out.println("Parsing operation succesfull for '" + args[0] + "' file");

} catch(Exception e) {

System.out.println("Parsing operation has failed ");

} } }

En este caso el objeto CoolScanner (scanner) toma el archivo a analizar y el objeto CoolParser usa la instancia de CoolScanner para iniciar el proceso de análisis sintáctico a la par con el análisis léxico.

Finalmente se parse todo el programa usando el método parse del parser y en caso de no arrojarse ninguna excepción se considera el análisis como exitoso.

Hay un par de líneas comentadas que no se pudieron implementar y que se relacionan con el análisis semántico.

### Generación del parser

Para generar el parser usando java\_cup en Linux el siguiente comando es usado.

java java\_cup.Main -parser CoolParser < cool.cup

Una vez generado se debe compilar el código resultante.

javac CoolParser.java

Después se genera el scanner.

jlex CoolScanner

Una vez generado se debe compilar el código resultante también.

javac CoolScanner.java

Por último se compila el código del archivo Main.java

javac Main.java

### Ejecución del scanner

Para que se ejecute el scanner solo se necesita ejecutar el siguiente comando

java Main $archivo

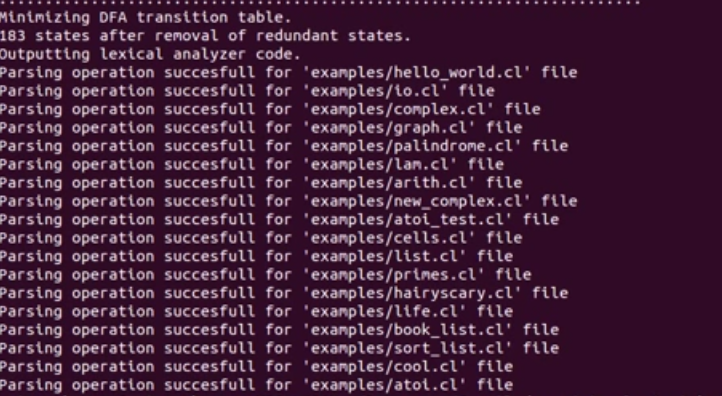
Donde $archivo es el nombre del archive a escanear. En el proyecto se provee un conjunto de archivos de prueba provistos por coursera.

Todos estos pasos son ejecutados automáticamente por el script autorun.sh.

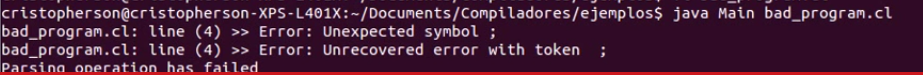
./autorun.sh

### Salida del programa

La salida del programa es un mensaje de éxito o fracaso dependiendo de si el archivo respeta la estructura del lenguaje especificado o un mensaje de error en caso de fallo.



En caso de error el siguiente mensaje es visto:



## Video de la ejecución del parser y scanner.

Para ver cómo funciona el programa final en un ambiente Linux se puede acceder al siguiente enlace de youtube:

<https://www.youtube.com/watch?v=8n2dN-PCLhU&feature=youtu.be>

# Conclusiones

Jlex y java\_cup son herramientas muy útiles para la generación de compiladores e implementación de lenguajes más potentes que los existentes o que cubran necesidades que otros no cubren. La ventaja que brinda es que el compilador generado en lenguaje java es portable en diferentes plataformas mientras haya una máquina virtual instalada en el dispositivo anfitrión.

En el caso de la implementación de un compilador del lenguaje Cool [8] se puedo generar un parser y un scanner que permita verificar que tanto el léxico usado como la gramática son funcionales. En una segunda etapa se implementará el analizador semántico para que el compilador esté más cerca de una etapa madura.

# Referencias

[1] <http://www.rogeliodavila.com/traduct/trnotes/intro.ppt>

[2] <http://web.stanford.edu/class/cs243/lectures/L1-handout.pdf>

[3] <http://www.cs.nuim.ie/~jpower/Courses/Previous/se209/inter/inter.pdf>

[4] <http://luv.asn.au/overheads/lex_yacc/lex.html>

[5] <http://luv.asn.au/overheads/lex_yacc/yacc.html>

[6] <https://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html>

[7] <http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>

[8] The Cool Reference Manual