# JLEX Y JAVA CUP

Jlex es una herramienta desarrollada en Java que toma como entrada un archivo “entrada”, con este crea un archivo fuente java entrada.lex.java correspondiente al analizador léxico. En otras palabras, Jlex es un generador de analizadores léxicos. Los analizadores léxicos toman como entrada una cadena de caracteres y la convierten en una secuencia de tokens.

JAVA CUP es un parser-generador. Es un analizador sintáctico que construye un parser para gramáticas tipo LALR(1), con código de producción y asociación de fragmentos de código JAVA. Cuando una producción en particular es reconocida, se genera un archivo fuente Java, parser.java que contiene una clase parser, con un método Symbol parser().

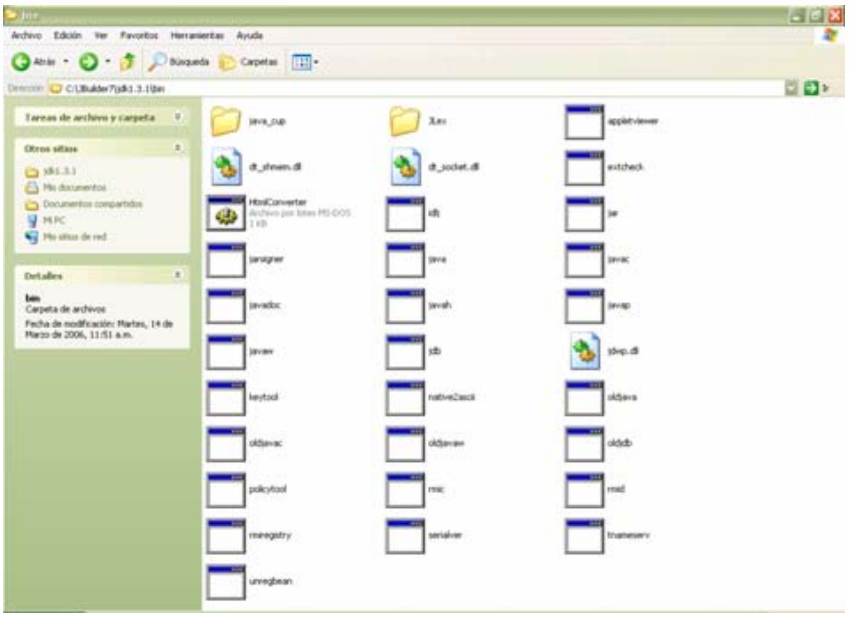
## Instalación

Para la instalación de Jlex y CUP se necesitan:

1. JDK (Descarga el JDK de la página Web de Sun para los sistemas Operativos Linux, Windows y Solaris). Si se tiene instalado alguna versión de JAVA, solo es necesario buscar el directorio en el que se encuentra el JDK de esta.
2. Archivo para la generación de las clases para Jlex llamado Main.java que se puede descargar de la página Web de Jlex.
3. Descarga del sitio Web de CUP el código fuente de los archivos necesarios.

Se debe crear una carpeta adentro de la carpeta bin del JDK, acá se creará una carpeta llamada Jlex y se incluirá en esta el archivo Main.java, a la vez adentro de la carpeta bin también se colocará la carpeta java_cup (Esta carpeta es la que descargamos del sitio CUP).

Por ejemplo: Si tenemos instalado el programa JBuilder7, esta carpeta se encuentra en la unidad C de nuestra computadora, adentro de esta carpeta esta el jdk.



A partir de acá generaremos nuestras variables de entorno que consiste en escribir el siguiente código desde una consola de DOS (Command Prompt)

1. Primero debemos colocarnos en la carpeta bin de la carpeta JDK desde la consola.

   En nuestro ejemplo seria así c:\jbuilder7\jdk1.3.1\bin

2. Ahora en la consola escribiremos

```
c:\jbuilder7\jdk1.3.1\bin> set CLASSPATH=C:\jbuilder7\jdk1.3.1\bin;%CLASSPATH%
c:\jbuilder7\jdk1.3.1\bin>set PATH=C:\jbuilder7\jdk1.3.1\bin;%PATH%
```

Con esto tenemos generadas las variables de entorno.

3. Ahora podemos compilar la clase Main.java. Para esto escribiremos en la consola

```
c:\jbuilder7\jdk1.3.1\bin>javac Jlex\Main.java
```

Con esto se generarán una serie de archivos adentro de la carpeta Jlex.
La carpeta de CUP no es necesario compilarla, ya que esta ya tiene todas las clases compiladas, ya solo se utilizan.

4. Ahora podemos crear nuestros archivos .lex y .cup. El archivo .lex será el archivo que tendrá los caracteres y las expresiones regulares válidas.

5. Ahora ya podemos empezar a utilizar Jlex y CUP, de primero creamos una carpeta donde realizaremos nuestro proyecto, por ejemplo en la unidad C. La carpeta se llamara PROYECTO, adentro de esta carpeta, copiaremos la carpeta de Jlex y la de Cup que copiamos de primero.

6. Se creará otra carpeta llamada Example, (Puede ser el nombre que se desee).

## CODIGO DE JLEX:

En un archivo de texto podemos escribir lo siguiente y guardarlo con extensión .lex

```
package Example;

import java_cup.runtime.Symbol;
%%
%cup
%%
";" {  //RECONOCE EL SIMBOLO PUNTO Y COMA
       return new Symbol(sym.SEMI); }
"+" { System.out.print("SIGNO DE SUMA ");//RECONOCE EL
SIMBOLO MAS
     return new Symbol(sym.PLUS); }
"*" {    System.out.print("SIGNO POR ");//RECONOCE EL
SIMBOLO POR
       return new Symbol(sym.TIMES); }
"-" {   System.out.print("SIGNO MENOS ");//RECONOCE EL
SIMBOLO MENOS
       return new Symbol(sym.MENOS); }
"/" {  System.out.print("SIGNO DIVIDIDO ");//RECONOCE EL
SIMBOLO DIVIDIDO
       return new Symbol(sym.DIVI); }
"(" { return new Symbol(sym.LPAREN); } //RECONOCE EL
PARENTESIS DE APERTURA
")" { return new Symbol(sym.RPAREN); }//RECONOCE EL
SIMBOLO PARENTESIS DE CIERRE
[0-9]+ { System.out.print(" numero ");//RECONOCE LOS
NUMEROS
       return new Symbol(sym.NUMBER, new
Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

Como observamos el package Example, es el archivo donde se generarán una serie de archivos que necesita el programa para trabajar como un analizador léxico, en todo caso lo importante es tener una carpeta con el nombre que tiene el package.

El import java_cup.runtime.Symbol se utiliza para importar los símbolos que se declararán en CUP, por tanto, cuando compilemos este archivo debemos tener el archivo de CUP ya

creado también. Los signos como punto y coma, mas, menos, por, dividido, paréntesis; son los signos que la gramática reconocerá; el System.out.print lo escribimos para que cuando los encuentre nos muestre en la consola la palabra que tenemos escrita entre comillas, como en el caso de suma nos desplegará en pantalla "SIGNO DE SUMA", y el nombre con el que lo guardará es con PLUS, que es para lo que se utiliza la instrucción return new Symbol(sym.PLUS); es decir, esta instrucción agrega a la tabla de símbolos el signo mas con el nombre de PLUS, en el caso de los números podemos observar que tenemos la siguiente instrucción return new Symbol(sym.NUMBER, new Integer(yytext())); que nos permitirá utilizar los números como texto, no solo como símbolos, por esto podemos utilizar ciertos atributos que nos permitirán manipularlos de cierta manera.

El código [\t\r\n\f] { /* ignore white space. */ } lo utilizamos para que ignore espacios en blanco, cambios de línea, tabulaciones, etc.

. { System.err.println("Illegal character: "+yytext()); } este código lo utilizamos en el caso que no encuentre el carácter que le decimos como cadena de entrada nos indique que hay un error, y pueda continuar con el análisis recuperando errores.

## CODIGO CUP

Archivo de texto guardado con extensión .cup

```
package Example;

import java_cup.runtime.*;
action code {:
/*CODIGO DE JAVA*/
:}
parser code {:
        public static void main(String args[]) throws Exception {
                new parser(new Yylex(System.in)).parse();
        }
:}

terminal SEMI, PLUS, TIMES, LPAREN, RPAREN;
terminal Integer NUMBER;

non terminal expr_list, expr_part;
non terminal Integer expr;

precedence left PLUS;
precedence left TIMES;

expr_list ::= expr_list expr_part | expr_part;
```

```
expr_part ::= expr:e {: System.out.println(" = "+e+";"); :} SEMI;
expr      ::= NUMBER:n
                {: RESULT=n; :}
              | expr:l PLUS expr:r
                {: RESULT=new Integer(l.intValue() + r.intValue()); :}
              | expr:l TIMES expr:r
                {: RESULT=new Integer(l.intValue() * r.intValue()); :}
              | LPAREN expr:e RPAREN
                {: RESULT=e; :}
              ;
```

Como podemos observar nuevamente en la carpeta package Example se crearán los archivos; importa la librería runtime de la carpeta de java_cup; en esta sección podemos colocar las librerías que necesitamos para poder compilar código de java; ya que en action code, podemos escribir código de java que necesitemos utilizar en nuestra acciones de la gramática. Donde se encuentran definidos los terminales, podemos ver que son los que definimos como símbolos en el archivo de Jlex, solo el Número que está declarado de tipo Integer para poder utilizarlo con los atributos de un número. En el caso que fuera una cadena, debemos declararlo de tipo String.

Los no terminales son los que utilizaremos en la gramática. Y después podemos declarar la precedencia si es necesario.

Supongamos que el archivo de texto se llama tok2.lex y el de cup se llama DOS3.cup.

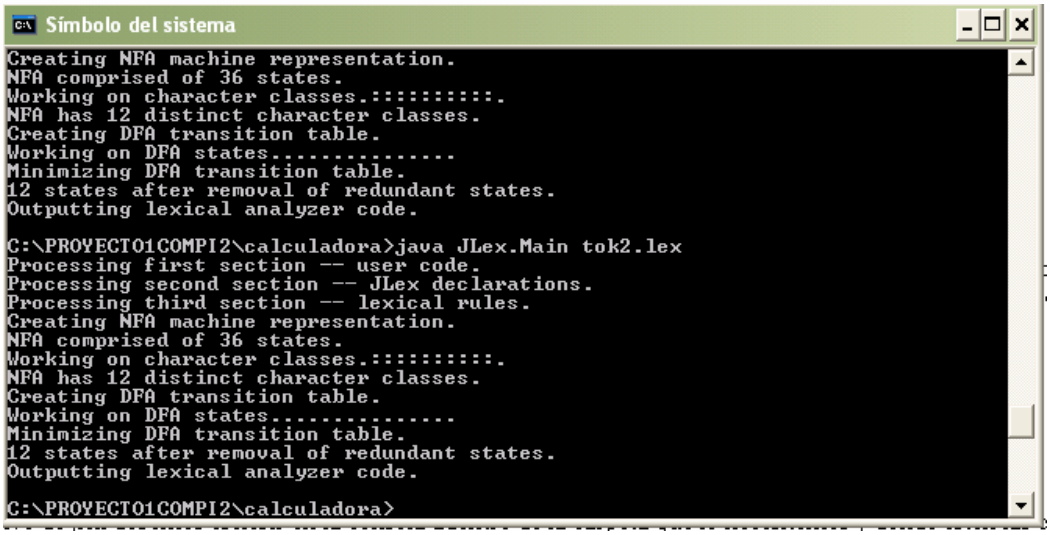
Ahora compilemos:

Para compilar el archivo de jlex debemos escribir en la consola adentro de la carpeta que lo necesitemos y donde estén las carpetas de JLex y cup

Mi proyecto está adentro de la carpeta PROYECTO1COMPI2\calculadora. Dentro de esta carpeta tengo la carpeta con las clases generadas de JLex y cup, la carpeta EXAMPLE y los archivos tok2.lex y DOS3.cup.

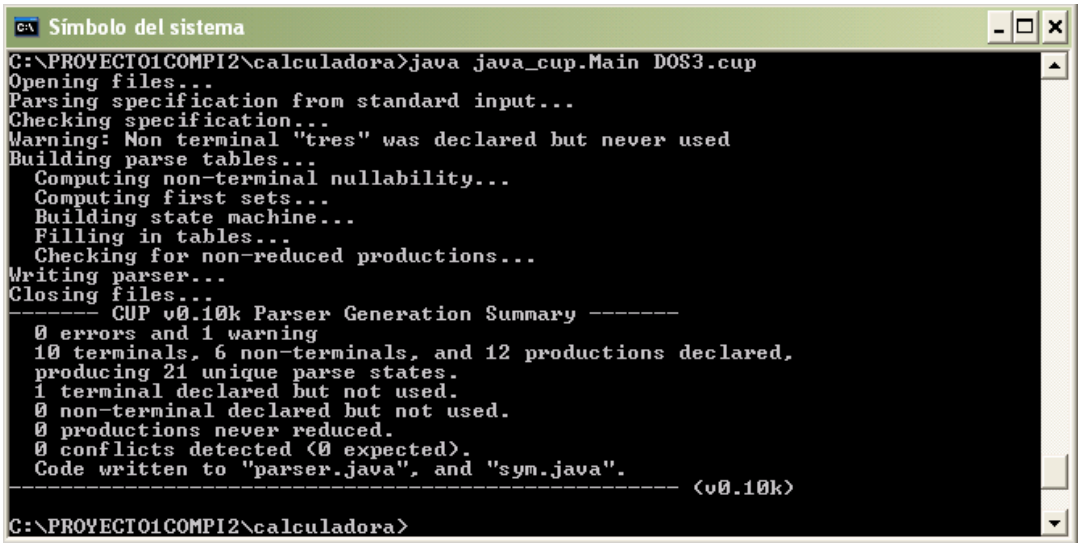
Escribo en la consola

```
java JLex.Main tok2.lex
```



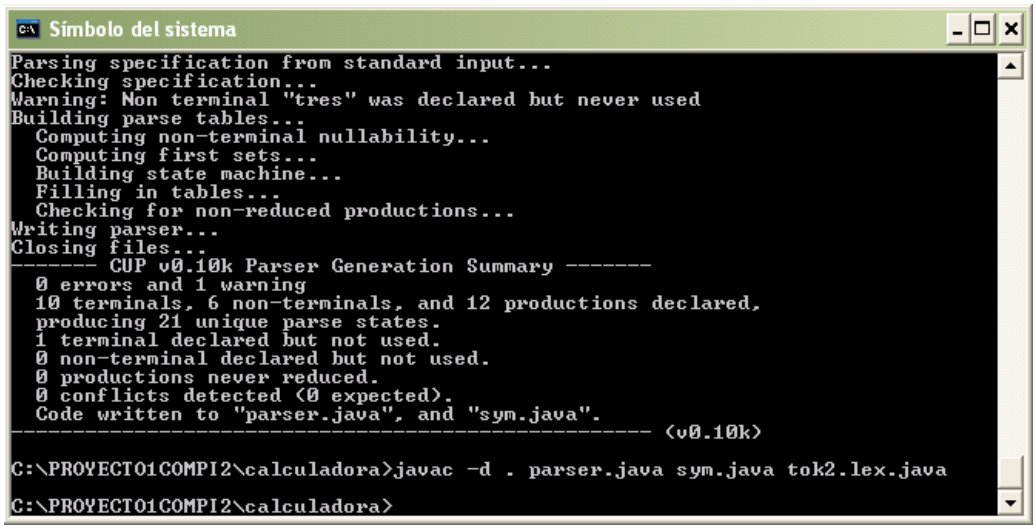

Obtendremos esa salida. Ahora compilemos el archivo de CUP

java java_cup.Main DOS3.cup




Ahora escribo

javac -d . parser.java sym.java tok2.lex.java




Ahora ya tenemos todas las clases necesarias. Podemos ya sea compilar un archivo de texto o podemos ingresarlo en la consola.

Por medio de un archivo de texto tenemos que escribir lo siguiente

```
java Example.parser 0<c:\\operar.txt
```

Y así en nuestro archivo de entrada c:\\operar.txt tenemos lo siguiente

```
5+2*3;
5+4+3;
```

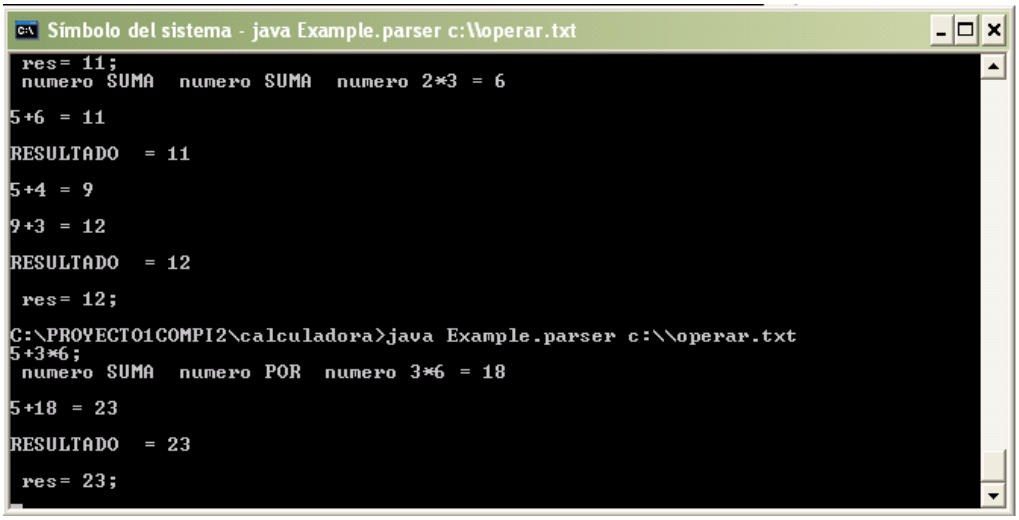
Y la salida es


Si ahora se escribe

java Example.parser c:\\operar.txt

Sin el 0<; podemos escribir las cadenas en la consola

Por ejemplo escribimos 5+3*6;




Ahora ya tenemos los archivos.

Como nos damos cuenta, esto corresponde al código de una calculadora, para esto utilizamos pilas dinámicas, por tanto el código es el siguiente:

## CODIGO tok2.lex

```
package Example;

import java_cup.runtime.Symbol;
%%
%cup
%%
";" {  //RECONOCE EL SIMBOLO PUNTO Y COMA
        return new Symbol(sym.SEMI); }
"+" { System.out.print("SUMA ");//RECONOCE EL SIMBOLO MAS
     return new Symbol(sym.PLUS); }
"*" {  System.out.print("POR ");//RECONOCE EL SIMBOLO POR
        return new Symbol(sym.TIMES); }
"-" {  System.out.print("MENOS ");//RECONOCE EL SIMBOLO MENOS
        return new Symbol(sym.MENOS); }
"/" {  System.out.print("DIVI ");//RECONOCE EL SIMBOLO DIVIDIDO
        return new Symbol(sym.DIVI); }
"(" { return new Symbol(sym.LPAREN); } //RECONOCE EL PARENTESIS
DE APERTURA
```

```
")" { return new Symbol(sym.RPAREN); }//RECONOCE EL SIMBOLO
PARENTESIS DE CIERRE
[0-9]+ { System.out.print(" numero ");//RECONOCE LOS NUMEROS
        return new Symbol(sym.NUMBER, new Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

CODIGO DOS3.cup

```
package Example;
/*declaracion librerias*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.io.File;
import java_cup.runtime.*;
import java.lang.ExceptionInInitializerError;
import java.io.IOException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;

/*CODIGO JAVA*/

action code {:
  public class nodomat{
     public nodomat(String cad)
     {
       cadenas=cad;
      // result=res;
       sig=null;
     }
    public String cadenas;
    //public int result=0;
    public nodomat sig;
  }
```

```
public class listamat{
  public nodomat inicio = null;
  public nodomat sigui = null;

  public void insertar(String c)
  {
   if (inicio==null)
   {
     inicio= new nodomat(c);
   }
   else
   {
   sigui=inicio;
    while(sigui.sig!=null)
     {
        sigui=sigui.sig;
     }
     sigui.sig =new nodomat(c);
   }

  }/*end void insertar*/

  public void ini()
  {
  try{
  FileWriter fw7 = new FileWriter("c:\\OPERACIONES.txt");
     BufferedWriter bw7 = new BufferedWriter(fw7);
     PrintWriter salida7 = new PrintWriter(bw7);
          salida7.close();
    }
     catch(java.io.IOException ioex) { }
  }

  public void recorrer() // para acciones
  {
try { //crea de categorías
     FileWriter fw7 = new FileWriter("c:\\OPERACIONES.txt",true);
      BufferedWriter bw7 = new BufferedWriter(fw7);
      PrintWriter salida7 = new PrintWriter(bw7);

    sigui=inicio;
     while(sigui!=null)
      { salida7.println(sigui.cadenas);
        System.out.println(sigui.cadenas+"\n ");
        sigui=sigui.sig;
      }
      salida7.close();
```

```
        }
         catch(java.io.IOException ioex) { }

      }/*end void recorrer*/
      }
   public listamat lm = new listamat();
:};
parser code {:
        public static void main(String args[]) throws Exception {
                new parser(new Yylex(System.in)).parse();
        }
        public void syntax_error(Symbol s)
        {int valor=0;
         valor=s.left+1;
          report_error("\nERROR SINTACTICO EN LINEA: "'+ (String)
s.value+"' línea: ("+valor+")\n",null);

          try {
FileWriter fw2 = new FileWriter("c:\\ERRORES_SINTAC_SEMAN.txt",true);
      BufferedWriter bw2 = new BufferedWriter(fw2);
      PrintWriter salida2 = new PrintWriter(bw2);
     salida2.println("\nERROR SINTACTICO EN LINEA: " + (String)
s.value+"' línea: ("+valor+")\n");
      salida2.close();
}
catch(java.io.IOException ioex) { }
        }

public void unrecovered_syntax_error(Symbol s) throws
  java.lang.Exception {
  report_fatal_error("", null);
}

:};

terminal SEMI, PLUS, TIMES, LPAREN, RPAREN, MENOS,DIVI;
terminal Integer NUMBER;

non terminal expr_list, expr_part,tres;
non terminal Integer expr;

precedence left PLUS, MENOS;
precedence left TIMES, DIVI;

expr_list ::= expr_list expr_part | expr_part;
expr_part    ::=    expr:e    {:lm.ini();lm.insertar("RESULTADO        =    "
+e);lm.recorrer(); System.out.println(" res= "+e+";");:} SEMI ;
```

```
expr      ::= NUMBER:n
                {: RESULT=n;  :}
            | expr:l PLUS:p expr:r
                {: RESULT=new Integer(l.intValue() + r.intValue()); lm.insertar( l
+ "+" + r + " = " + RESULT);  :}
                | expr:l MENOS:m expr:r
                {: RESULT=new Integer(l.intValue() - r.intValue());lm.insertar( l +
"-" + r+ " = " + RESULT); :}
                | expr:l TIMES:t expr:r
                {: RESULT=new Integer(l.intValue() * r.intValue());lm.insertar( l +
"*" + r+ " = " + RESULT); :}
                | expr:l DIVI:t expr:r
                {: RESULT=new Integer(l.intValue() / r.intValue());lm.insertar( l +
"/" + r+ " = " + RESULT); :}
                | LPAREN expr:e RPAREN
                {: RESULT=e; :}
                |error
                ;
```