

Autores:

Leonel Alberto Madrid Franco	1190-11-8687
Jorge Antonio Laj Guerra	1190-11-9902
Abdiel David Fajardo Armas	1190-11-3464
Carla Mariza Cacao Cabrera	1190-11-2575
Delmer Aníbal Picen Orellana	1390-10-5688

COMPILADOR XML

COMPILADOR DISEÑADO PARA GRAFICAR

Este compilador esta diseñado para Graficar Líneas, Círculos, Rectángulos y Cambiarles de Color, El Compilador esta diseñado con dos Herramientas muy útiles de Java, Analizador Léxico JFlex y el Analizador Sintáctico Cup.

COMPILADOR XML

Jlex:

Jlex es una herramienta desarrollada en Java que usaremos para realizar el analizador léxico de nuestro compilador. Las instrucciones de instalación son muy sencillas, y son las mismas para cualquier Sistema Operativo:

1. Crea un directorio nuevo (aquí lo llamaremos jdir) que esté en tu CLASSPATH (también puedes usar un directorio que ya exista). Crea un directorio llamado jdir/JLex (o jdir\JLex en Windows), y copia en el fichero Main.java que podrás descargar de la página web4 de Jlex.

2. Compila el fichero Java con el siguiente comando:

```
javac Main.java
```

3. Se habrán creado las clases Java que componen la distribución de Jlex.

Ahora ya podrás usar el Jlex mediante este comando:

```
java Jlex.Main fichero.jlex
```

donde fichero.jlex es el nombre del fichero con la especificación del análisis sintáctico para tu lenguaje.

Alternativamente a este proceso de instalación, los usuarios de Linux podrán instalar un

paquete binario RPM o DEB con la distribución de Jlex, lista para usar. En concreto, los usuarios de

Debian podrán obtener el paquete del FTP oficial del proyecto Debian y podrán obtenerlos

mediante este comando:

```
apt-get install jlex
```

para usar esta distribución de Jlex se debe utilizar este comando:

```
jlex fichero.jlex
```

donde fichero.jlex tiene el mismo significado que antes. En este caso no será necesario modificar el CLASSPATH.

CUP:

CUP es la herramienta que usaremos para generar el analizador sintáctico de nuestro

lenguaje. Al igual que JLex está escrita en Java, y existe un proceso de instalación común para todas

las plataformas:

1. Descarga el código fuente de CUP desde su página web5 a un directorio de tu sistema que

esté en el CLASSPATH. Descomprime el paquete una vez lo hayas descargado

2. Ahora compilaremos el código fuente de CUP. Para ello, desde el directorio donde

descargamos la distribución, ejecutaremos el siguiente comando:

```
javac java_cup/*.java java_cup/runtime/*.java
```

Ahora podrás ejecutar CUP mediante el siguiente comando:

```
java java_cup.Main < fichero.cup
```

donde fichero.cup es el fichero con la especificación del analizador sintáctico de nuestro lenguaje.

Y ya está hecho. Si eres usuario de Linux, de nuevo puedes saltarte estas instrucciones e instalarte el paquete binario para tu distribución. En este caso, podrás ejecutar CUP mediante este comando:

```
cup < fichero.cup
```

· Jasmin:

Jasmin es un ensamblador para la Máquina Virtual Java, el cual toma como entrada un

fichero de texto con la descripción de una clase Java, y produce como resultado el .class de esa

clase. Nosotros usaremos esta herramienta para obtener el programa ejecutable resultado del

proceso de compilación.

El proceso de instalación de Jazmín se puede resumir en los siguientes pasos:

1. Descarga la distribución de su página web⁶, y descomprímelo en /usr/local si eres usuario

de UNIX, o C:\ si eres usuario de Windows. Se creará un directorio llamado jasmin.

2. Ahora debes añadir el directorio /usr/local/jasmin/bin (o C:\jasmin\bin) a tu variable de

entorno \$PATH. En UNIX bastará con hacer un enlace simbólico a /usr/local/bin, si es

que este directorio está ya en el \$PATH:

```
ln -s /usr/local/jasmin/bin/jasmin /usr/local/bin/
```

Ahora podrás ejecutar Jazmín mediante el siguiente comando:

```
jazmín fichero.j
```

Donde fichero.j es la descripción de una clase Java en ensamblador.

NOTA: en algunas sistemas Windows, tendrás que añadir el directorio C:\jasmin\classes a tu CLASSPATH para que funcione correctamente.

Configuraciones

Analizador Léxico

```
1  /* primera parte: no hace falta poner nada */
2  package clasesPrograma;
3  import java_cup.runtime.Symbol;
4  %%
5  /* segunda parte: declaramos las directivas y los macros */
6  %class AnalizadorLexico
7  %public
8  %full
9  %unicode
10 %line
11 %column
12 %char
13 %cup
14
15 LineTerminator = \r|\n|\r\n|\n\r
16 WhiteSpace = {LineTerminator} | [ \t\f]
17 ValorEntero = 0|[1-9][0-9]*
18 Comentario = [//] [a-z][a-z]*[0-9]*
19
20 %%
21 /* OPERADORES Y SIGNOS */
22 "</" {return new Symbol(sym.PCOMA, new token(yycolumn, yyline, yytext()));}
23 ", " {return new Symbol(sym.COMA, new token(yycolumn, yyline, yytext()));}
24 "(" {return new Symbol(sym.ABRIRPAR, new token(yycolumn, yyline, yytext()));}
25 ")" {return new Symbol(sym.CERRARPAR, new token(yycolumn, yyline, yytext()));}
26
27 /* PALABRAS RESERVADAS */
28 "<XML>" {return new Symbol(sym.INICIO, new token(yycolumn, yyline, yytext()));}
29 "</XML>" {return new Symbol(sym.FIN, new token(yycolumn, yyline, yytext()));}
30 "<circulo>" {return new Symbol(sym.CIRCULO, new token(yycolumn, yyline, yytext()));}
31 "<linea>" {return new Symbol(sym.LINEA, new token(yycolumn, yyline, yytext()));}
32 "<rectangulo>" {return new Symbol(sym.RECTANGULO, new token(yycolumn, yyline, yytext()));}
33 "<color>" {return new Symbol(sym.COLOR, new token(yycolumn, yyline, yytext()));}
34
35 /* EXPRESIONES */
36 {ValorEntero} {return new Symbol(sym.VALINT, new token(yycolumn, yyline, yytext()));}
37 {LineTerminator} {return new Symbol(sym.ENTER, new token(yycolumn, yyline, yytext()));}
38 {WhiteSpace} {/* ignorar */}
39 . {System.err.println("caracter invalido" + yytext() + "[" + yyline + ":" + yycolumn + "]);}
```


Analizador Sintáctico

```
1 package clasesPrograma;
2
3 import java_cup.runtime.*;
4 import java.util.ArrayList;
5
6
7 action code {:
8     ArrayList<Instruccion> instrucciones = new ArrayList();
9
10 :}
11
12 parser code {:
13     //esta es la manera en que se puede acceder a un objeto que se genera durante la etapa del parsing
14     public ArrayList<Instruccion> getInstrucciones(){
15         return action_obj.instrucciones;
16     }
17
18     @Override
19     public void syntax_error(Symbol sy) {
20         token t=(token)sy.value;
21         done_parsing();
22         report_error("Error sintáctico cerca de \""+ t.getCadena()+"\" : ["+t.getRow()+" : "+t.getCol()+"",null);
23     }
24 :}
25
26
27 /*-----TERMINALES-----*/
28
29 terminal COMA,PCOMA,ABRIRPAR,CERRARPAR,INICIO,FIN,RECTANGULO,CIRCULO,LINEA,ENTER,COLOR;
30 terminal token VALINT;
31
32 /*-----NO TERMINALES-----*/
33
34 non terminal instrucciones, programa;
35 non terminal Instruccion instruccion,<linea>,<circulo>,<rectangulo>,<color>;
36
37 /*-----GRAMATICAS-----*/
38 start with programa;
39
40 /*-----ESTRUCTURA DE PROGRAMA-----*/
41 programa ::= INICIO ENTER instrucciones ENTER FIN;
42
43 /*-----DEFINICION DE INSTRUCCIONES-----*/
44 instrucciones ::= instrucciones ENTER instruccion| instruccion;
45 instruccion ::= circulo|color|linea|rectangulo;
46 linea ::= LINEA ABRIRPAR VALINT:a COMA VALINT:b COMA VALINT:c COMA VALINT:d CERRARPAR PCOMA LINEA
47     {:int[] tmp = new int[4];
48     tmp[0]=Integer.parseInt(a.getCadena());
49     tmp[1]=Integer.parseInt(b.getCadena());
50     tmp[2]=Integer.parseInt(c.getCadena());
51     tmp[3]=Integer.parseInt(d.getCadena());
52     instrucciones.add(new Instruccion("linea",tmp));:};
53 rectangulo ::= RECTANGULO ABRIRPAR VALINT:a COMA VALINT:b COMA VALINT:c COMA VALINT:d CERRARPAR PCOMA RECTANGULO
54     {:int[] tmp = new int[4];
55     tmp[0]=Integer.parseInt(a.getCadena());
56     tmp[1]=Integer.parseInt(b.getCadena());
57     tmp[2]=Integer.parseInt(c.getCadena());
58     tmp[3]=Integer.parseInt(d.getCadena());
59     instrucciones.add(new Instruccion("rectangulo",tmp));:};
60 color ::= COLOR ABRIRPAR VALINT:a COMA VALINT:b COMA VALINT:c CERRARPAR PCOMA COLOR
61     {:int[] tmp = new int[3];
62     tmp[0]=Integer.parseInt(a.getCadena());
63     tmp[1]=Integer.parseInt(b.getCadena());
64     tmp[2]=Integer.parseInt(c.getCadena());
65     instrucciones.add(new Instruccion("color",tmp));:};
66 circulo ::= CIRCULO ABRIRPAR VALINT:a COMA VALINT:b COMA VALINT:c CERRARPAR PCOMA CIRCULO
67     {:int[] tmp = new int[3];
68     tmp[0]=Integer.parseInt(a.getCadena());
69     tmp[1]=Integer.parseInt(b.getCadena());
70     tmp[2]=Integer.parseInt(c.getCadena());
71     instrucciones.add(new Instruccion("circulo",tmp));:};
```

Función del COMPILADORXML

la función del COMPILADORXML es de graficar, líneas, círculos, rectángulos y colorearlos.

Para apertura del programa utilizamos.

```
<XML>
```

Para Finalización del Programa.

```
</XML>
```

Para ponerle Color a las Formas

```
<color>( 0, 0, 0)</color>
```

Para Graficar un Círculo

```
<circulo>( X, Y, RADIO)</circulo>
```

Para Graficar una Línea

```
<linea>( X, Y, LARGO,ANGULO)</linea>
```

Para Graficar un Rectángulo

```
<rectangulo>( X, Y, ANCHO, ALTO)</rectangulo>
```

Ejemplo utilizando todos los recursos.

```
<XML>
<color>(33,25,200)</color>
<circulo>(10,10,100)</circulo>
<circulo>(40,10,100)</circulo>
<circulo>(70,10,100)</circulo>
<circulo>(100,10,100)</circulo>
<linea>(10,120,200,120)</linea>
<rectangulo>(5,5,200,250)</rectangulo>
<color>(33,255,200)</color>
<circulo>(10,140,100)</circulo>
<circulo>(40,140,100)</circulo>
<circulo>(70,140,100)</circulo>
<circulo>(100,140,100)</circulo>
</XML>
```

Le damos ejecutar y el resultado es

