2021

MANUAL TÉCNICO

CALCULADORA JFLEX

LENOVO

Universidad Da Vinci de Guatemala

Facultad de Ingeniería

Ingeniería en Sistemas

Compiladores

Ingeniero Rubén López

Kevyn Salvador Posadas Tote 201901858
Maria Rene Esteban García 201801482

CALCULADORA JFLEX

Fecha

28/04/2021


Java

Calculadora con JFLEX y CUP

Herramientas

Netbeans

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

JFLex y CUP

son dos herramientas que generan programas que reaccionen a una entrada de datos con una estructura y un lenguaje predeterminado. Como ejemplo se pueden crear compiladores intérprete y analizadores de línea de comando, dando una opción a la solvencia necesitada.

- JLEX

Jlex no es más que un generador de un analizador léxico parecido a LEX, el cual toma una cadena como entrada una cadena de caracteres, y lo convierte en una secuencia de t okens.

- CUP

Cup es un generador de analizadores sintácticos LALR en Java el cual recibe de entrada un archivo con la estructura de la gramática y su salida es un parser escrito en Java listo para usarse.

Realización de la calculadora

A_Lexer

Dentro de nuestro proyecto hacemos una carpeta llamada analizadores, dentro de ella creamos un empty file llamado A_lexer.cup, en donde estarán nuestras gramáticas

Colocamos el paquete en donde se creará el archivo y hacemos las importaciones de las librerías.

```
/*------------  1ra Area: Codigo de Usuario ---------*/
//------> Paquetes,importaciones
package Analizadores;
import java_cup.runtime.*;
import java.util.LinkedList;
```

En el **área número 2** hacemos las declaraciones, declaramos una estructura en donde se guardarán los errores y colocamos las directivas de Jflex.

```
/*------------  2da Area: Opciones y Declaraciones ---------*/
%%
%{
    //----> Codigo de usuario en sintaxis java
    public static LinkedList<TError> TablaEL = new LinkedList<TError>();
%}

//-------> Directivas
%public
%class Analizador_Lexico
%cupsym Simbolos
%cup
%char
%column
%full
%ignorecase
%line
%unicode
```

Por último, colocamos nuestras expresiones regulares, en donde D representa una expresión regular que reconoce números enteros y DD una expresión regular que reconoce números con punto decimal.

```
//------> Expresiones Regulares
numero = [0-9]+ // expresion regular para validar entero
numeroD=[0-9]+("."[  |0-9]+)? // expresion regular para validar decimales
```

En la tercera área estarás las reglas léxicas, en donde indicaremos el patrón que se va a reconocer y adentro de las llaves se coloca lo que se debe hacer cuando se reconozca. En Símbolos ER, en donde será similar a lo anterior, pero en este caso llamamos a nuestras expresiones regulares. En Espacios ignoraremos los espacios en blanco.

```
%%
/*------------  3raa Area: Reglas Lexicas ---------*/

//-----> Simbolos

<YYINITIAL> "+"         { System.out.println("Reconocio "+yytext()+" mas"); return new Symbol(Simbolos.mas, yycolumn, yyline, yytext()); }
<YYINITIAL> "-"         { System.out.println("Reconocio "+yytext()+" menos"); return new Symbol(Simbolos.menos, yycolumn, yyline, yytext()); }
<YYINITIAL> "*"         { System.out.println("Reconocio "+yytext()+" por"); return new Symbol(Simbolos.por, yycolumn, yyline, yytext()); }
<YYINITIAL> "/"         { System.out.println("Reconocio "+yytext()+" div"); return new Symbol(Simbolos.div, yycolumn, yyline, yytext()); }
<YYINITIAL> "("         { System.out.println("Reconocio "+yytext()+" para"); return new Symbol(Simbolos.para, yycolumn, yyline, yytext()); }
<YYINITIAL> ")"         { System.out.println("Reconocio "+yytext()+" parc"); return new Symbol(Simbolos.parc, yycolumn, yyline, yytext()); }

//-------> Simbolos ER para validar los numeros.
<YYINITIAL> {numero}    { System.out.println("Reconocio "+yytext()+" num"); return new Symbol(Simbolos.num, yycolumn, yyline, yytext()); }

<YYINITIAL> {numeroD}    { System.out.println("Reconocio "+yytext()+" numD"); return new Symbol(Simbolos.numD, yycolumn, yyline, yytext()); }

//------> Espacios
[ \t\r\n\f]             {/* Espacios en blanco, se ignoran */}
```

En errores léxicos cualquier símbolo que no sean los que declaramos anteriormente será tomado como error, entonces nos mostrará la línea y columna en donde se encontró el error. Si queremos guardar los errores en una lita enlazada creamos un método público static parametrizada con un bojeto, el cual se llamará TError, y la lista enlazada lleva el nombre de TablaEL **(Errores léxicos)** la cuál debe ser importada en el área 1.

```
//------> Errores Lexicos
.                       { System.out.println("Error Lexico"+yytext()+" Linea "+yyline+" Columna "+yycolumn);
                          TError datos = new TError(yytext(),yyline,yycolumn,"Error Lexico","Simbolo no existe en el lenguaje");
                          TablaEL.add(datos);}
```

TError

Creamos el objeto de tipo de errores TError, en donde haremos un string para lexema, tipo y descripción y dos variables de tipo entero las cuales serán línea y columna. Haremos un constructor público al cual le pasaremos los datos.

Una vez hecho esto regresamos al leer y guardamos los errores, creando un nuevo objeto llamado TError y le pasamos los datos al constructor.

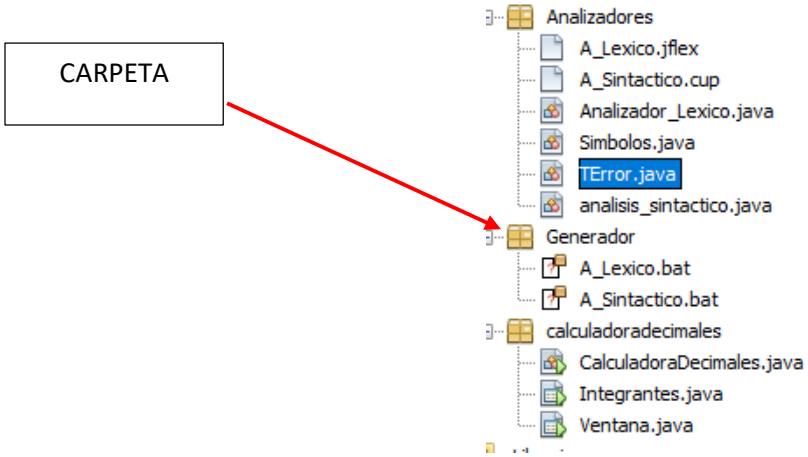
```
 * and open the template in the editor.
 */
package Analizadores;

/**
 *
 * @author David
 */
public class TError {
    String lexema,tipo,descripcion;
    int linea,columna;

    public TError(String le, int li, int co, String t,String de){
        lexema = le;
        linea = li;
        columna = co;
        tipo = t;
        descripcion = de;
    }

```

Hacemos un nuevo paquete llamado **Generadores** y creamos un archivo llamado A_lexico.bat en donde se encontrarán las rutas de las librerías



A_Sintactico

Este archivo vamos a incluir el código que le dirá a Cup que es lo que debe hacer.

En **el área número 1** se importarán los paquetes y librerías, luego en “parser code”, se programan acciones propias del parser o analizador sintáctico que se va a genera, hacemos una variable de salida tipo Double.

```
/*---------------- 1ra Area: Codigo de Usuario ----------------------*/
//-------> importaciones, paquetes
package Analizadores;
import java_cup.runtime.Symbol;
import java.util.LinkedList;

//------> Codigo para el parser,variables, metodos
parser code
{:
    public Double resultado=0.0; // variable de salida tipo Double
    public static LinkedList<TError> TablaES = new LinkedList<TError>();
```

Creamos un método el cuál se llamará automáticamente cuando se encuentre algún error sintáctico y llamamos a la lista que guarda los errores.

```
//Metodo al que se llama automaticamente ante algun error sintactico
public void syntax_error(Symbol s)
{
    String lexema = s.value.toString();
    int fila = s.right;
    int columna = s.left;

    System.out.println("!!!!!!! Error Sintactico Recuperado !!!!!!!");
    System.out.println("\t\tLexema: "+lexema);
    System.out.println("\t\tFila: "+fila);
    System.out.println("\t\tColumna: "+columna);

    TError datos = new TError(lexema,fila,columna,"Error Sintactico","Caracter no esperado");
    TablaES.add(datos);
```

Luego hacemos otro método al que se llama en el momento en que ya no sea posible una recuperación de errores

```
//Metodo al que se llama en el momento en que ya no es posible una recuperacion de errores
public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
{
    String lexema = s.value.toString();
    int fila = s.right;
    int columna = s.left;

    System.out.println("!!!!!!! Error Sintactico, Panic Mode !!!!!!! ");
    System.out.println("\t\tLexema: "+lexema);
    System.out.println("\t\tFila: "+fila);
    System.out.println("\t\tColumna: "+columna);

    TError datos = new TError(lexema,fila,columna,"Error Sintactico","Caracter no esperado");
    TablaES.add(datos);

}
```

En el **área número dos** definimos los terminales, a estos se les puede indicar un tipo, en este c aso todos son de tipo *String,* si no se indicara un tipo, los terminales serían por defecto de tipo *Object*.

```
/*---------------- 2da Area: Declaraciones ------------------------*/
//------> declaracion de terminales

terminal String para,parc;
terminal String mas,menos,por,div;
terminal String num; // el valor de entrada del numero entero lo tomamos como String
terminal String numD; // el valor de entrada del numero decimal lo tomamos como String
```

Luego declaramos los no terminales, a los que también se les puede indicar un tipo específico, la expresión la fijamos como double.

```
//------> declaracion de no terminales
non terminal INICIO;
non terminal Double E; // la expresion la fijamos como double
```

Procedemos a indicar la precedencia de los operadores, es decir, el orden de las operaciones, la precedencia más baja la tienen la suma y la resta, luego están la multiplicación y la división que tienen una precedencia más alta y por último está el signo menos de las expresiones negativas que tendría la precedencia más alta.

```
//----> precedencia de menor a mayor
precedence left mas,menos;
precedence left por,div;


start with INICIO;
```

Por último, se colocan las reglas de producción, al fin de cada producción, al final de cada producción puede incluirse código javo entre llaves y dos puntos “{:<código java>:}”. Podemos ver que en las producciones del no terminal “expresion”, se utiliza la variable RESULT, esta variable es propia de Cup y nos permite sintetizar cierto atributo para ese no terminal que se encuentra del lado izquierdo de la producción, RESULT puede ser cualquier objeto.

```
/*--------------- 3ra Area: Reglas Semanticas ----------------------*/
INICIO::=E:a {: resultado=a; :};

E  ::= // el resultado que sale en la expresion es de tipo double
    E:a mas        E:b     {:RESULT=a+b;:}
  | E:a menos      E:b     {:RESULT=a-b;:}
  | E:a por        E:b     {:RESULT=a*b;:}
  | E:a div        E:b     {:RESULT=a/b;:}
  | num:a                          {:RESULT=new Double(a);:}
  | numD:a                         {:R ESULT=new Double(a);:}
  | para E:a parc             {:RESULT=a;:}
  |
;
```

Una vez hecho esto creamos el archivo sintactico.bat para poder compilarlos.

Ventana:

Botones

Creamos un string global para cuando se oprima un botón se pueda imprimir el valor que se asigna, para que guarde el valor se hace una concatenación con todo lo que está en el string

```
private void btnNumero1MouseClicked(java.awt.event.MouseEvent evt) {
    System.out.println("El valor es:" + "1");
    String valor1 = "1";
    concatenar = concatenar.concat(valor1);
    txtEntrada.setText(concatenar);
}
```

Resultado Final