Analizador Sintáctico Parte I

Analizador Léxico para CUP

Conforme a la metodología de desarrollo del JFlex y JCUP se deben de desarrollar los siguientes códigos.

En el “Lexer.flex” realiza los siguientes ajustes:

1. Identifica a las palabras reservadas por separado.
2. Utiliza el método de lexemas para trabajar con todos los símbolos.
3. Agrega a los componentes léxicos el salto de línea.
4. Agrega paréntesis, llaves, punto y coma, main.

```
package codigo;
import static codigo.Tokens.*;
%%
%class Lexer
%type Tokens
L=[a-zA-Z_]+
D=[0-9]+
espacio=[ ,\t,\r]+
%{
    public String lexeme;
%}
%%
"int" {lexeme=yytext(); return Int;}
"if" {lexeme=yytext(); return If;}
"else" {lexeme=yytext(); return Else;}
"while" {lexeme=yytext(); return Reservadas;}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}
"\n" {lexeme=yytext(); return Linea;}
"=" {lexeme=yytext(); return Igual;}
```

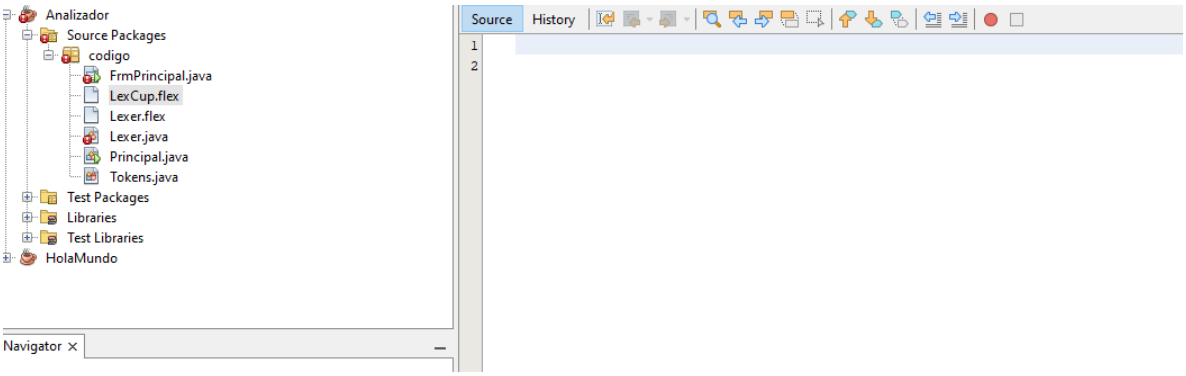
```
"+" {lexeme=yytext(); return Suma;}
"-" {lexeme=yytext(); return Resta;}
"*" {lexeme=yytext(); return Multiplicacion;}
"/" {lexeme=yytext(); return Division;}
"(" {lexeme=yytext(); return Parentesis_a;}
")" {lexeme=yytext(); return Parentesis_c;}
"{" {lexeme=yytext(); return Llave_a;}
"}" {lexeme=yytext(); return Llave_c;}
"main" {lexeme=yytext(); return Main;}
";" {lexeme=yytext(); return Punto_c;}
{L}({L}|{D})* {lexeme=yytext(); return Identificador;}
("(-"{D}+")")|{D}+|{D}+"."{D}* {lexeme=yytext(); return Numero;}
 . {return ERROR;}
```

En el “Token.java” agrega los tokens que faltan:

```
public enum Tokens {
    Int,
    If,
    Else,
    While,
    Linea,
    Igual,
    Suma,
    Resta,
    Multiplicacion,
    Division,
    Parentesis_a,
    Parentesis_c,
    Llave_a,
    Llave_c,
    Main,
    Punto_c,
    Identificador,
    Numero,
    ERROR
```

*NOTA: no preocupes si marca error en tus archivos, se corregirán cuando se realicen todos los ajustes requeridos.

Crea el LexerCup.flex, en el Codigo crea un nuevo archivo (New-Other-Other-Empty File)



Lo que hay en el Lexer.flex se copia al nuevo archivo.

Para el Analizador Sintáctico no se requiere tener el salto de línea por separado. Se vuelve agregar en el conjunto “espacio”.

Se cambia la línea del import por:

```
import java_cup.runtime.Symbol;
```

El type cambia por:

```
%type java_cup.runtime.Symbol
%cup
%full
%line
%char
```

Los cuales van a permitir: **cup** realizar el análisis, **full** recordar la cadena completa, **line** la línea en la que aparece, **char** para el número de carácter.

La cadena global “public String lexeme;” cambia por la siguiente sobrecarga, que va a permitir configurar los parámetros a ser considerados durante el análisis sintáctico.

```
%{
    private Symbol symbol(int type, Object value){
        return new Symbol (type, yyline, yycolumn, value);
    }
    private Symbol symbol(int type){
        return new Symbol (type, yyline, yycolumn);
    }
%}
```

Deberá de quedar de la siguiente manera:

```
package codigo;
import java_cup.runtime.Symbol;
%%
%class Lexer
%type java_cup.runtime.Symbol
%cup
%full
%line
%char
L=[a-zA-Z_]+
D=[0-9]+
espacio=[ ,\t,\r,\n]+
%{
    private Symbol symbol(int type, Object value){
        return new Symbol (type, yyline, yycolumn, value);
    }
    private Symbol symbol(int type){
        return new Symbol (type, yyline, yycolumn);
    }
%}
%%
```

Se tendrán que ajustar todos los lexemas para que generen terminales útiles para el analizador sintáctico, en JCUP a las terminales se les conoce como símbolo. En este contexto lo que interesa es número de carácter y la línea en la que se encuentra el terminal. Para hacer los ajustes considera lo siguiente:

int {return new Symbol (Sym.Int, yychar, yyline, yytext());}

Realiza el proceso para todas las terminales, quedando de la siguiente manera:

```
"int" {return new Symbol (Sym.Int, yychar, yyline, yytext());}
"if" {return new Symbol (Sym.If, yychar, yyline, yytext());}
"else" {return new Symbol (Sym.Else, yychar, yyline, yytext());}
"while" {return new Symbol (Sym.While, yychar, yyline, yytext());}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}
"=" {return new Symbol (Sym.Igual, yychar, yyline, yytext());}
"+" {return new Symbol (Sym.Suma, yychar, yyline, yytext());}
"-" {return new Symbol (Sym.Resta, yychar, yyline, yytext());}
"*" {return new Symbol (Sym.Multiplicacion, yychar, yyline, yytext());;}
"/" {return new Symbol (Sym.Division, yychar, yyline, yytext());}
"(" {return new Symbol (Sym.Parentesis_a, yychar, yyline, yytext());}
")" {return new Symbol (Sym.Parentesis_c, yychar, yyline, yytext());;}
"{" {return new Symbol (Sym.Llave_a, yychar, yyline, yytext());;}
"}" {return new Symbol (Sym.Llave_c, yychar, yyline, yytext());}
"main" {{return new Symbol (Sym.Main, yychar, yyline, yytext());}
";" {{return new Symbol (Sym.Punto_c, yychar, yyline, yytext());}
{L}({L}|{D})* {{return new Symbol (Sym.Identificador, yychar, yyline, yytext());}
("(-"{D}+")")|{D}+|{D}+"."{D}* {{return new Symbol (Sym.Numero, yychar, yyline, yytext());;}
 . {{return new Symbol (Sym.Error, yychar, yyline, yytext());}
```

No olvidar guardar todo lo trabajado…

Para las siguientes clases:

1. Analizador Sintáctico para CUP
2. Clase para generar ambos archivos