Compiladores 2025-1

Profesora: Ariel Adara Mercado Martínez
Ayudante: Carlos Gerardo Acosta Herández

Práctica 1: Analizadores léxicos con Lex (JFlex)

Nombre	**No. de Cuenta**
Vázquez Torrijos Damián	318309877

29 de Agosto del 2024

Introducción

Lex es una herramienta para generar analizadores léxicos, que se deben describir mediante las expresiones regulares de los tokens que serán reconocidas por el analizador léxico (scanner o lexer). Originalmente fue desarrollado para el sistema operativo Unix, pero con la popularidad de Linux se creo una versión para este sistema llamada Flex.

Estructura de un archivo Lex

Un programa en LEX consta de tres secciones:

```
Sección de declaraciones
%%
Sección de expresiones regulares
%%
Sección de código de usuario (código en lenguaje C++)
```

Sección de declaraciones

- **Directivas de código.** Se utilizan para incluir los archivos de biblioteca y definir las variables globales es la siguiente:

```
%{
    int contador;
%}
```

- **Macros o definiciones.** Las macros son variables a las que se les asigna una expresión regular. Por ejemplos, `letra[a-zA-Z]`, la macro se llama *letra*, separada por un espacio de la definición de su expresión regular.
- **Directivas de Lex.** Las directivas u opciones de escáner le indican a Lex que realice tareas extras al momento de generar el analizador léxico.
  - `%public` Hace pública la clase que genera.
  - `%class` Le indica a JFlex qué nombre tendrá la clase generada.
  - `%state` o `%xstate` es para declarar estados léxicos.

Sección de expresiones regulares

- **Estructura:** `<Expresión Regular>={<Acción Léxica>}`
  - **Expresión Regular:** Debe estar escrita con la sintaxis de Lex.

- **Acción Léxica:** Se ejecuta cada vez que se encuentra una cadena que coincida con la expresión regular y es escrita en lenguaje Java, encerrada por llaves.

Metacaracteres

Caracter	Descripción
c	Cualquier carácter representado por c que no sea un operador
\c	El carácter c literalmente
"S"	La cadena s, literalmente
.	Cualquier carácter excepto el salto de línea
^	Inicio de línea
$	Fin de línea
[s]	Cualquier carácter incluido dentro de la cadena s
[^s]	Cualquier carácter que no esté dentro de s
\n	Salto de línea
*	Cerradura de Kleene
+	Cerradura positiva
|	Disyunción
?	Cero o una instancia
{m, n}	Entre m y n instancias de una expresión que le antecede

Sección de código de usuario

En esta sección se escriben las funciones auxiliares para realizar el análisis léxico, por lo general es donde se agrega a main.

Desarrollo

Una vez instalado flex, se prosiguió a transcribir el código proporcionado a un archivo con extensión `.flex` dentro de la carpeta `/src/Primer_programa_en_Lex`.

archivo.flex

src > Primer_programa_en_Lex > archivo.flex

```
/**
* Escáner que detecta números y palabras
*/

%%

%public
%class Lexer
%standalone

digito=[0-9]
letra=[a-zA-Z]
palabra={letra}+
espacio=[ \t\n]

%%

{espacio}+  /* La acción léxica puede ir vacía si queremos que el
escáner ignore la regla */
{digito}+ { System.out.println("Encontré un número: " + yytext()+"\n"); }
{palabra} { System.out.println("Encontré una palabra: " + yytext()+"\n"); }
```

Se continuó con la compilación y verificación de la existencia del archivo `Lexer.java`

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>jflex archivo.flex
Reading "archivo.flex"
Constructing NFA : 16 states in NFA
Converting NFA to DFA :
.......
9 states before minimization, 5 states in minimized DFA
Writing code to "Lexer.java"

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>dir
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: 94CF-4E28

 Directorio de C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex

26/08/2024  04:06 p. m.    <DIR>          .
26/08/2024  04:06 p. m.    <DIR>          ..
26/08/2024  04:03 p. m.               427 archivo.flex
26/08/2024  04:06 p. m.            21,480 Lexer.java
               2 archivos         21,907 bytes
               2 dirs  56,372,350,976 bytes libres

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

Después se compiló a `Lexer.java`

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>javac Lexer.java

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

Se creó un archivo de texto con la entrada `"Hay 100 nubes en el cielo hoy"`

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>echo "Hay 100 nubes en el cielo hoy" > input.txt

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

Finalmente se ejecutó todo mediante el comando `java Lexer input.txt`

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>java Lexer input.txt
"Encontré una palabra: Hay

Encontré un número: 100

Encontré una palabra: nubes

Encontré una palabra: en

Encontré una palabra: el

Encontré una palabra: cielo

Encontré una palabra: hoy

"
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

Ejercicios.

1. **¿Qué ocurre si en la primera sección se quitan las llaves al nombre de la macro letra?**

   En un principio, no se obtienen errores al momento de compilar la modificación.

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>jflex archivo.flex
Reading "archivo.flex"
Constructing NFA : 26 states in NFA
Converting NFA to DFA :
............
14 states before minimization, 11 states in minimized DFA
Old file "Lexer.java" saved as "Lexer.java~"
Writing code to "Lexer.java"

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

   Pero al momento de hacer la ejecución con `java Lexer input.txt`, se puede notar que solo reconoce la expresión regular del macro *digito*.

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>javac Lexer.java

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>java Lexer input.txt
"HayEncontré un número: 100

nubesenelcielohoy"
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>
```

2. **¿Qué pasa si en la segunda sección se quitan las llaves a las macros**

   En este caso no se puede compilar debido a errores de sintaxis.

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>jflex archivo.flex
Reading "archivo.flex"

Error in file "archivo.flex" (line 20):
Syntax error.
{digito}+  System.out.println("Encontré un número: " + yytext()+"\n");
                                                      ^


Error in file "archivo.flex" (line 21):
Syntax error.
{palabra}  System.out.println("Encontré una palabra: " + yytext()+"\n");
                                                        ^

2 errors, 0 warnings.
```

3. **¿Cómo se escribe un comentario en flex?**

   Casi, si no que de la misma manera en Java: `/* */` cuando son varias líneas o `//` cuando es una línea.

4. **¿Qué se guarda en yytext?**

   Guarda la cadena que coincide con la expresión regular.

5. **¿Qué pasa al ejecutar el programa e introducir cadenas de caracteres y de dígitos por las consola?**

   Se tuvieron problemas para implementar un método Main para que pudiera pasarle cadenas directamente desde la terminal, por lo que se creó un `input2.txt`. Los resultados al final fueron que los número dentro de las cadenas no fueron reconocidos como *digitos* y se tomaron como espacios.

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>echo "UnaCadenacon678numeros A1" > input2.txt

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>java Lexer input2.txt
"Encontré una palabra: UnaCadenacon

678Encontré una palabra: numeros

 Encontré una palabra: A

1"
```

6. **¿Qué ocurre si introducimos caracteres como ”*” en la consola?**

   Lo mismo que en la pregunta anterior, los nuevos caracteres no fueron reconocidos.

```
C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>echo "Una cadena con 2 **" > input3.txt

C:\Users\carfar\Documents\DamianVT\Prácticas\P1\src\Primer_programa_en_Lex>java Lexer input3.txt
"Encontré una palabra: Una

 Encontré una palabra: cadena

 Encontré una palabra: con

Encontré un número:  2

 **"
```

7. **Modificar al código anterior en un archivo nuevo, de tal manera que reconozca lo siguiente:**

   a) **La expresión regular para los hexadecimales en lenguaje Java.**

   b) **5 palabras reservadas del lenguaje Java.**

   c) **Los identificadores válidos del lenguaje Java, con longitud máxima de 32 caracteres**

   d) **Los espacios en blanco.**

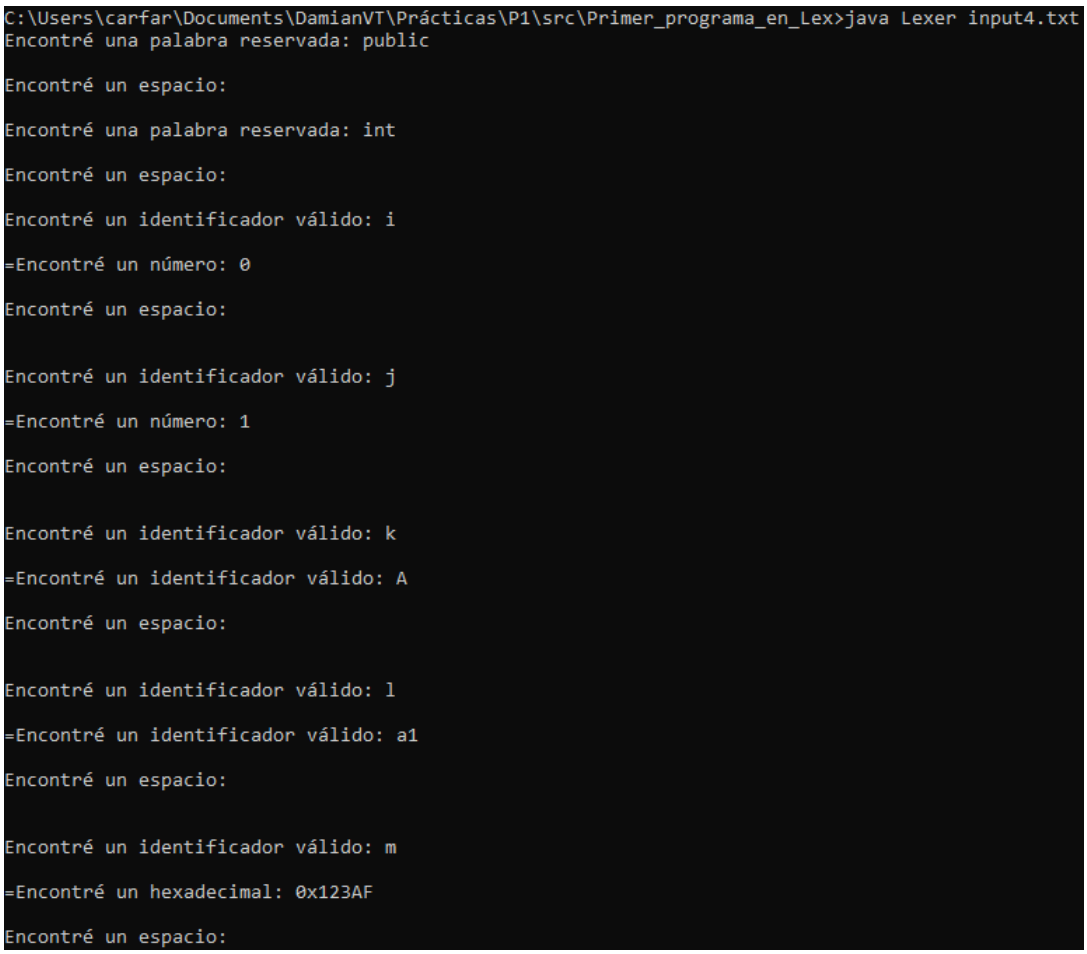
```
/**
* Escáner que detecta números y palabras
*/

%%

%public
%class Lexer
%standalone
digito=[0-9]
letra=[a-zA-Z]
hexa =[0-9a-fA-F]
p_reservadas=("public"|"private"|"protected"|"static"|"final"|"void"|"int")
identificador = {letra}({letra}|{digito}|_){0,31}
espacio=[ \t\n]
hexadecimal=0[xX]{hexa}+

%%

{espacio}+  { System.out.println("Encontré un espacio: " + yytext()+"\n"); }
{hexadecimal} { System.out.println("Encontré un hexadecimal: " + yytext()+"\n"); }
{digito}+ { System.out.println("Encontré un número: " + yytext()+"\n"); }
{p_reservadas} { System.out.println("Encontré una palabra reservada: " + yytext()+"\n");
{identificador} { System.out.println("Encontré un identificador válido: " + yytext()+"\n
{letra}+ { System.out.println("Encontré una palabra: " + yytext()+"\n"); }
```



Conclusión

Se pudo conocer y utilizar los principios para generar analizadores léxicos utilizando Lex. Estuvo entretenido, aunque si hay que estar muy atentos al orden de las definiciones.