AUTÓMATAS Y COMPILADORES

Dr. Eduardo Cornejo -Velázquez

NOMBRE DE LA PRÁCTICA: PRÁCTICA INICIO CON LEX

ALUMNO:

Daniel Martínez Escamilla

Introducción

En el presente reporte se documenta la realización de seis ejercicios utilizando FLEX, un generador de analizadores léxicos empleado en el procesamiento de autómatas y compiladores. Este software permite definir patrones mediante expresiones regulares para identificar y categorizar elementos dentro de un flujo de entrada, facilitando así la primera etapa del análisis sintáctico en el desarrollo de lenguajes de programación. A lo largo de la práctica, se explorarán las funcionalidades básicas y avanzadas de FLEX, implementando reglas léxicas para reconocer estructuras específicas dentro de un archivo. Cada ejercicio abordará diferentes aspectos del análisis léxico, desde la identificación de números hasta la identificación de palabras reservadas. El objetivo principal es comprender el funcionamiento de FLEX, su sintaxis y su integración con herramientas como YACC, lo que resulta fundamental en el diseño de compiladores y procesadores de lenguaje.

Herramientas empleadas

FLEX (Fast Lexical Analyzer Generator) es una herramienta que permite la generación automática de analizadores léxicos a partir de un conjunto de reglas definidas por el usuario. Su principal función es leer un archivo de entrada y clasificar fragmentos del texto según patrones especificados mediante expresiones regulares, generando código en C que puede integrarse con otros programas, como analizadores sintácticos creados con YACC.
Esta herramienta es ampliamente utilizada en el desarrollo de compiladores e intérpretes de lenguajes de programación, ya que facilita la segmentación y reconocimiento de tokens en la etapa inicial del procesamiento del código fuente. FLEX ofrece una estructura flexible y eficiente, lo que permite su aplicación en diversas tareas relacionadas con el análisis de texto, como procesamiento de logs, extracción de datos y filtrado de contenido.

Test 1

Reconocer números enteros

```
%option noyywrap
%{
        #include<stdio.h>
        #define NUMEROS_ENTEROS 1

%}

%%
1 {return NUMEROS_ENTEROS; }
2 {return NUMEROS_ENTEROS; }
3 {return NUMEROS_ENTEROS; }
4 {return NUMEROS_ENTEROS; }
5 {return NUMEROS_ENTEROS; }
6 {return NUMEROS_ENTEROS; }
7 {return NUMEROS_ENTEROS; }
8 {return NUMEROS_ENTEROS; }
9 {return NUMEROS_ENTEROS; }
0 {return NUMEROS_ENTEROS; }
%%
int main()
{
        int token;
        while(1){
        token = yylex();
        if ( token == NUMEROS_ENTEROS ){
              printf("  token NUMEROS_ENTEROS reconocido\n");}


        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 1: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test1.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test1

C:\Flex Windows\EditPlusPortable>test1.exe
```

Figura 2: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test1.exe
3
  token NUMEROS_ENTEROS reconocido

4
  token NUMEROS_ENTEROS reconocido

34
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido

123456789
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
  token NUMEROS_ENTEROS reconocido
```

Figura 3: Probando en Terminal

Test 1 opción 2

Reconocer números enteros

```
%option noyywrap
%{
        #include<stdio.h>
        #define NUMEROS_ENTEROS 1

%}

%%
[+-]?[0-9]+ {return NUMEROS_ENTEROS; }
%%
int main()
{
        int token;
        while(1){
        token = yylex();
        if ( token == NUMEROS_ENTEROS ){
              printf("  token NUMEROS_ENTEROS reconocido\n");}

        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 4: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test1.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test1

C:\Flex Windows\EditPlusPortable>test1.exe
```

Figura 5: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test1.exe
1
  token NUMEROS_ENTEROS reconocido

+1
  token NUMEROS_ENTEROS reconocido

-2
  token NUMEROS_ENTEROS reconocido

+99
  token NUMEROS_ENTEROS reconocido

-566
  token NUMEROS_ENTEROS reconocido
```

Figura 6: Probando en Terminal

Test 2

Reconocer números decimales

```
1  %option noyywrap
2  %{
3          #include<stdio.h>
4          #define NUMEROS_DECIMALES 1
5  %}
6
7  %%
8  [+-]?[0-9]+([.][0-9]+)? {return NUMEROS_DECIMALES; }
9  %%
10 int main()
11 {
12 int token;
13         while(1){
14         token = yylex();
15         if ( token == NUMEROS_DECIMALES ){
16               printf("  token NUMEROS_DECIMALES reconocido\n");}
17
18
19         if ( token == 0 )
20             break;
21         }
22         return 0;
23 }
```

Figura 7: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test2.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test2

C:\Flex Windows\EditPlusPortable>test2.exe
```

Figura 8: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test2.exe
1
  token NUMEROS_DECIMALES reconocido

1.67
  token NUMEROS_DECIMALES reconocido

+4.234
  token NUMEROS_DECIMALES reconocido

-567.356
  token NUMEROS_DECIMALES reconocido
```

Figura 9: Probando en Terminal

Test 3

Reconocer variables (identificadores). Inician con una letra y despúes puede tener letra, números y guiones bajos 5

```
%option noyywrap
%{
        #include<stdio.h>
        #define VARIABLE 1
%}

%%
[a-z]([0-9a-zA-Z_]+)? {return VARIABLE; }
%%

int main()
{
int token;
        while(1){
        token = yylex();
        if ( token == VARIABLE ){
              printf("  token VARIABLE reconocido\n");}


        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 10: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test3.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test3

C:\Flex Windows\EditPlusPortable>test3.exe
```

Figura 11: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test3.exe
1
1
a
  token VARIABLE reconocido

a_
  token VARIABLE reconocido

A
A
aa
  token VARIABLE reconocido

asdcAAASDF___AAA
  token VARIABLE reconocido

z
  token VARIABLE reconocido
```

Figura 12: Probando en Terminal

Test 4

Reconocer RFC:

```
%option noyywrap
%{
        #include<stdio.h>
        #define RFC 1
%}

%%
[A-Z]{4}[0-9]{2}([0]+[1-9]{1}|1[0-2])([0]+[1-9]{1}|1[1-9]|2[1-9]|3[0-1])[0-9A-Z]{3} {return RFC; }
%%
int main()
{
int token;
        while(1){
        token = yylex();
        if ( token == RFC ){
              printf("  token RFC reconocido\n");}


        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 13: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test4.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test4

C:\Flex Windows\EditPlusPortable>test4.exe
```

Figura 14: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test4.exe
MAED040813M2H
  token RFC reconocido

MAED4813M2H
MAED4813M2H
XEXT990101NI4
  token RFC reconocido

XEXT9911NI4
XEXT9911NI4
```

Figura 15: Probando en Terminal

Test 5

Reconocer CURP:

```
%option noyywrap
%{
        #include<stdio.h>
        #define CURP 1
%}

%%
[A-Z]{4}[0-9]{2}([0]+[1-9]{1}|1[0-2])([0]+[1-9]{1}|1[0-9]|2[0-9]|3[0-1])(H|M)[A-Z]{5}[A-Z0-9]{2} {return CURP; }
%%
int main()
{
int token;
        while(1){
        token = yylex();
        if ( token == CURP ){
              printf("  token CURP reconocido\n");}


        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 16: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test5.l

C:\Flex Windows\EditPlusPortable>cc lex.yy.c -o test5

C:\Flex Windows\EditPlusPortable>test5.exe
```

Figura 17: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test5.exe
 SARM781210MPBLZN09
   token CURP reconocido

GOLA850315MJLNNN05
  token CURP reconocido

PEPJ900101HDFRRN09
  token CURP reconocido
```

Figura 18: Probando en Terminal

Test 6

Reconocer email institucional:

```
%option noyywrap
%{
        #include<stdio.h>
        #define EMAIL_INSTITUCIONAL 1
%}

%%
[a-z]{2}[0-9]{6}[@](uaeh)[.](edu)[.](mx) {return EMAIL_INSTITUCIONAL; }
%%
int main()
{
int token;
        while(1){
        token = yylex();
        if ( token == EMAIL_INSTITUCIONAL ){
              printf("  token EMAIL_INSTITUCIONAL reconocido\n");}


        if ( token == 0 )
            break;
        }
        return 0;
}
```

Figura 19: Sintaxis en FLEX

Compilando y Ejecutando el programa :

```
C:\Flex Windows\EditPlusPortable>lex test6.l

C:\Flex Windows\EditPlusPortable>cc  lex.yy.c -o test6

C:\Flex Windows\EditPlusPortable>test6.exe
```

Figura 20: Compilacion en Terminal

Probando el programa:

```
C:\Flex Windows\EditPlusPortable>test6.exe
ma428615@uaeh.edu.mx
  token EMAIL_INSTITUCIONAL reconocido

ma428615@uaeh.edumx
ma428615@uaeh.edumx
ma428615uaeh.edu.mx
ma428615uaeh.edu.mx
ma428615uae.edu.mx
ma428615uae.edu.mx
m428615uae.edu.mx
m428615uae.edu.mx
```

Figura 21: Probando en Terminal

Conclusiones

Esta práctica permitió comprender cómo las herramientas de análisis léxico, como FLEX, se utilizan en el desarrollo de compiladores y procesadores de lenguajes para validar patrones de texto específicos. Al finalizar, se pudo verificar el funcionamiento adecuado del analizador mediante pruebas, lo que garantiza que el código es capaz de reconocer el formato de números enteros, con decimales, formatos complejos como CURP, correos, RFC. Este tipo de herramientas y técnicas es esencial en el desarrollo de aplicaciones que requieran validaciones específicas y eficientes en grandes volúmenes de datos.

Referencias

[1] El Análisis Léxico en yacc: flex. (n.d.).
https://campusvirtual.ull.es/ocw/pluginfile.php/2311/mod$_r esource/content/0/perlexamples/node216.html$