Instituto Politécnico Nacional

Escuela Superior De Computo

Práctica:

Práctica 4

Alumno:

Fernando Vizcaíno López

Boleta:

2017631591

Carrera Docente:

ISC

Introducción

Flex es una herramienta que genera un analizador léxico. A partir de un conjunto de expresiones regulares, Flex busca concordancias en un fichero de entrada y ejecuta acciones asociadas a estas expresiones. Es compatible con Lex, una herramienta clásica de Unix para la generación de analizadores léxicos, pero es un desarrollo diferente realizado por GNU bajo licencia GPL.

El proyecto por desarrollar será hacer un compilador en C, recibirá como entada un código fuente y como salida entregará código ensamblador, esto con el propósito entre a los lenguajes de alto nivel y una máquina. Permitiendo desarrollar y programar de una forma más precisa los lenguajes de alto nivel.

El GCC es un compilador integrado del proyecto GNU que al principio solo compilaba el lenguaje C. Posteriormente se extendió para compilar C++, Fortran, Ada y otros. Es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.

El desarrollo de C se llevó a cabo en los Laboratorios Bell de AT&T entre 1969 y 1973; según Ritchie, el periodo más creativo tuvo lugar en 1972. Se le dio el nombre "C" porque muchas de sus características fueron tomadas de un lenguaje anterior llamado "B".

Se opto por este compilador en C porque en un principio se tenía contemplado hacer otro lenguaje que recibiera un programa fuente, similar al del lenguaje C y tendría como salida la estructura de un árbol. El compilador que se tenía contemplado en un principio consistía en programar una seria de preguntas, respuestas y variables. Donde cada pregunta y respuesta que se programa podría programar una seria de acciones de lo que quiera que suceda y es así como se avanzaría al siguiente nodo de la estructura del árbol. Lo que se tenía contemplado era programar tu propio juego con tus propias reglas y de lo que quieras que sucediera, en un lenguaje fuente, y la salida seria que te construyera un árbol o un autómata de ese juego.

Al final no se escogió el juego porque no estoy seguro si más adelante en el curso si podría desarrollarlo.

La siguiente versión de Flex que se está utilizando es la 2.6.4

Metodóloga

I. Ejemplificar el lenguaje

Ejemplo 1. Asignación de variables:

```
int var_int = 10;

float var_float = 32.124

char var_b = ‘F’
```

Ejemplo 2. Ciclos y operadores

```
int a = 0;

do {

        a += 1;

} while ( a < 3 );

for ( a = 0; i <= 3; i++ );
```

II. Identificar las clases léxicas

Las siguientes clases léxicas son las siguientes:

Tipo Dato, Sentencia Control, Comentario, Bucle, ID, Palabra Reservada, Hexadecimal, Constante, Cadena, Operador, Operador Asignación, Operador Lógico, Operador Condicional, Operador Relacional, Operador Aritmético, Operador bit bit, Separador, Agrupador y Saldo Línea

III. Escribir las expresiones para cada clase léxica

1.	D	[0-9]
2.	L	[a-zA-Z_]
3.	H	[a-fA-F0-9]
4.	E	[Ee][+-]?{D}+
5.	FS	(f|F|l|L)
6.	IS	(u|U|l|L)*
7.	"/*"	{ printf("<Comentario>"); }
8.	"char"	{ printf("<Tipo Dato>");
9.	"float"	{ printf("<Tipo Dato>");
10.	"double"	{ printf("<Tipo Dato>"); }
11.	"int"	{ printf("<Tipo Dato>"); }
12.	"long"	{ printf("<Tipo Dato>"); }
13.	"short"	{ printf("<Tipo Dato>"); }
14.	"if"	{ printf("<Sentencia Control>"); }
15.	"else"	{ printf("<Sentencia Control>"); }
16.	"switch"	{ printf("<Sentencia Control>"); }
17.	"case"	{ printf("<Sentencia Control>"); }
18.	"do"	{ printf("<Bucle>"); }
19.	"for"	{ printf("<Bucle>"); }
20.	"while"	{ printf("<Bucle>"); }
21.	"auto"	{ printf("<Palabra Reservada>"); }
22.	"break"	{ printf("<Palabra Reservada>"); }
23.	"const"	{ printf("<Palabra Reservada>"); }
24.	"continue"	{ printf("<Palabra Reservada>"); }
25.	"default"	{ printf("<Palabra Reservada>"); }
26.	"enum"	{ printf("<Palabra Reservada>"); }
27.	"extern"	{ printf("<Palabra Reservada>");
28.	"goto"	{ printf("<Palabra Reservada>"); }
29.	"register"	{ printf("<Palabra Reservada>"); }
30.	"return"	{ printf("<Palabra Reservada>"); }
31.	"signed"	{ printf("<Palabra Reservada>"); }
32.	"sizeof"	{ printf("<Palabra Reservada>"); }
33.	"static"	{ printf("<Palabra Reservada>"); }
34.	"struct"	{ printf("<Palabra Reservada>"); }
35.	"typedef"	{ printf("<Palabra Reservada>"); }
36.	"union"	{ printf("<Palabra Reservada>"); }
37.	"unsigned"	{ printf("<Palabra Reservada>"); }
38.	"void"	{ printf("<Palabra Reservada>"); }
39.	"volatile"	{ printf("<Palabra Reservada>"); }
40.	{L}({L}|{D})*	{ printf("<ID>"); }

```
41. 0[xX]{H}+{IS}?          { printf("<Hexadecimal>"); }
42. 0{D}+{IS}?             { printf("<Octal>"); }
43. {D}+{IS}?              { printf("<Decimal>"); }
44. L?'(\\.|[^\\'])+'      { printf("<Caracter>"); }
45. {D}+{E}{FS}?           { printf(" <Exponencial> "); }
46. {D}*"."{D}+({E})?{FS}? { printf(" <Real Esponencial> "); }
47. {D}+"."{D}*({E})?{FS}? { printf(" <Real Esponencial> "); }
48. L?\"(\\.|[^\\"])*\"    { printf(" <Cadena> "); }
49. "..."                  { printf(" <Operador> "); }
50. "--"                   { printf(" <Operador> "); }
51. "->"                   { printf(" <Operador> "); }
52. ">>="                  { printf(" <Operador Asignacion> "); }
53. "<<="                  { printf(" <Operador Asignacion> "); }
54. "="                    { printf(" <Operador Asignacion> "); }
55. "+="                   { printf(" <Operador Asignacion> ");
56. "-="                   { printf(" <Operador Asignacion> "); }
57. "*="                   { printf(" <Operador Asignacion> "); }
58. "/="                   { printf(" <Operador Asignacion> "); }
59. "%="                   { printf(" <Operador Asignacion> "); }
60. "&="                   { printf(" <Operador Asignacion> "); }
61. "^="                   { printf(" <Operador Asignacion> "); }
62. "|="                   { printf(" <Operador Asignacion> "); }
63. "++"                   { printf(" <Operador Asignacion> "); }
64. "&&"                   { printf(" <Operador Logico> "); }
65. "||"                   { printf(" <Operador Logico> "); }
66. "!"                    { printf(" <Operador Logico> "); }
67. "?"                    { printf(" <Operador Condicional> "); }
68. ":"                    { printf(" <Operador Condicional> "); }
69. "<="                   { printf(" <Operador Relacional> "); }
70. ">="                   { printf(" <Operador Relacional> "); }
71. "=="                   { printf(" <Operador Relacional> "); }
72. "!="                   { printf(" <Operador Relacional> "); }
73. "<"                    { printf(" <Operador Relacional> "); }
74. ">"                    { printf(" <Operador Relacional> "); }
75. "-"                    { printf(" <Operador Aritmetico> "); }
76. "+"                    { printf(" <Operador Aritmetico> "); }
77. "*"                    { printf(" <Operador Aritmetico> "); }
78. "/"                    { printf(" <Operador Aritmetico> "); }
79. "%"                    { printf(" <Operador Aritmetico> "); }
80. "&"                    { printf(" <Operador Bit A Bit> "); }
81. "^"                    { printf(" <Operador Bit A Bit> "); }
82. "|"                    { printf(" <Operador Bit A Bit> "); }
83. "~"                    { printf(" <Operador Bit A Bit> "); }
84. ">>"                   { printf(" <Operador Bit A Bit> "); }
85. "<<"                   { printf(" <Operador Bit A Bit> "); }
86. ";"                    { printf(" <Separador> "); }
87. ","                    { printf(" <Separador> "); }
88. ("{"|"<%")             { printf(" <Agrupador> "); }
89. ("}"|"%>")             { printf(" <Agrupador> "); }
```

```
90. "("                 { printf(" <Agrupador> "); }
91. ")"                 { printf(" <Agrupador> "); }
92. ("["|"<:")          { printf(" <Agrupador> "); }
93. ("]"|":>")          { printf(" <Agrupador> "); }
94. "."                 { printf(" <Agrupador> "); }
95. [ \t\v\n\f]         { printf("\n"); }
96. .                   { /* ignore bad characters */ }
```

IV. Codificar en Lex

```
D           [0-9]
L           [a-zA-Z_]
H           [a-fA-F0-9]
E           [Ee][+-]?{D}+
FS          (f|F|l|L)
IS          (u|U|l|L)*

%{
#include <stdio.h>
%}

%%
"/*"            { printf("<Comentario>"); }

"char"          { printf("<Tipo Dato>"); }
"float"         { printf("<Tipo Dato>"); }
"double"        { printf("<Tipo Dato>"); }
"int"           { printf("<Tipo Dato>"); }
"long"          { printf("<Tipo Dato>"); }
"short"         { printf("<Tipo Dato>"); }

"if"            { printf("<Sentencia Control>"); }
"else"          { printf("<Sentencia Control>"); }
"switch"        { printf("<Sentencia Control>"); }
"case"          { printf("<Sentencia Control>"); }

"do"            { printf("<Bucle>"); }
"for"           { printf("<Bucle>"); }
"while"         { printf("<Bucle>"); }
```

```
"auto"          { printf("<Palabra Reservada>"); }
"break"         { printf("<Palabra Reservada>"); }
"const"         { printf("<Palabra Reservada>"); }
"continue"      { printf("<Palabra Reservada>"); }
"default"       { printf("<Palabra Reservada>"); }
"enum"          { printf("<Palabra Reservada>"); }
"extern"        { printf("<Palabra Reservada>"); }
"goto"          { printf("<Palabra Reservada>"); }
"register"      { printf("<Palabra Reservada>"); }
"return"        { printf("<Palabra Reservada>"); }
"signed"        { printf("<Palabra Reservada>"); }
"sizeof"        { printf("<Palabra Reservada>"); }
"static"        { printf("<Palabra Reservada>"); }
"struct"        { printf("<Palabra Reservada>"); }
"typedef"       { printf("<Palabra Reservada>"); }
"union"         { printf("<Palabra Reservada>"); }
"unsigned"      { printf("<Palabra Reservada>"); }
"void"          { printf("<Palabra Reservada>"); }
"volatile"      { printf("<Palabra Reservada>"); }


{L}({L}|{D})*       { printf("<ID>"); }

0[xX]{H}+{IS}?      { printf("<Hexadecimal>"); }
0{D}+{IS}?      { printf("<Octal>"); }
{D}+{IS}?       { printf("<Decimal>"); }
L?'(\\.|[^\\'])+'   { printf("<Caracter>"); }
```

```
L?\"(\\.|[^\\"])*\" { printf(" <Cadena> "); }

"..."           { printf(" <Operador> "); }
"--"            { printf(" <Operador> "); }
"->"            { printf(" <Operador> "); }

">>="           { printf(" <Operador Asignacion> "); }
"<<="           { printf(" <Operador Asignacion> "); }
"="         { printf(" <Operador Asignacion> "); }
"+="            { printf(" <Operador Asignacion> "); }
"-="            { printf(" <Operador Asignacion> "); }
"*="            { printf(" <Operador Asignacion> "); }
"/="            { printf(" <Operador Asignacion> "); }
"%="            { printf(" <Operador Asignacion> "); }
"&="            { printf(" <Operador Asignacion> "); }
"^="            { printf(" <Operador Asignacion> "); }
"|="            { printf(" <Operador Asignacion> "); }
"++"            { printf(" <Operador Asignacion> "); }

"&&"            { printf(" <Operador Logico> "); }
"||"            { printf(" <Operador Logico> "); }
"!"         { printf(" <Operador Logico> "); }

"?"         { printf(" <Operador Condicional> "); }
":"         { printf(" <Operador Condicional> "); }
```

```
"<="            { printf(" <Operador Relacional> "); }
">="            { printf(" <Operador Relacional> "); }
"=="            { printf(" <Operador Relacional> "); }
"!="            { printf(" <Operador Relacional> "); }
"<"         { printf(" <Operador Relacional> "); }
">"         { printf(" <Operador Relacional> "); }

"-"         { printf(" <Operador Aritmetico> "); }
"+"         { printf(" <Operador Aritmetico> "); }
"*"         { printf(" <Operador Aritmetico> "); }
"/"         { printf(" <Operador Aritmetico> "); }
"%"         { printf(" <Operador Aritmetico> "); }

"&"         { printf(" <Operador Bit A Bit> "); }
"^"         { printf(" <Operador Bit A Bit> "); }
"|"         { printf(" <Operador Bit A Bit> "); }
"~"         { printf(" <Operador Bit A Bit> "); }
">>"            { printf(" <Operador Bit A Bit> "); }
"<<"            { printf(" <Operador Bit A Bit> "); }

";"         { printf(" <Separador> "); }
","         { printf(" <Separador> "); }

("{"|"<%")      { printf(" <Agrupador> "); }
("}"|"%>")      { printf(" <Agrupador> "); }
"("         { printf(" <Agrupador> "); }
")"         { printf(" <Agrupador> "); }
("["|"<:")      { printf(" <Agrupador> "); }
("]"|":>")      { printf(" <Agrupador> "); }
"."         { printf(" <Agrupador> "); }
```

```
[ \t\v\n\f]     { printf("\n"); }
.           { /* ignore bad characters */ }

%%
```

V. Pruebas

Conclusiones

Gracias a esta práctica pude aprender el uso de esta herramienta llamada Flex. La experiencia que tuve al usar Flex anteriormente en la primera clase de compiladores y esta fue poner a prueba los conocimientos de las expresiones regulares. La investigación y demostración de uso de la herramienta Flex resultó satisfactoria exponiendo en ella los resultados esperados presentando con ejemplos para un mayor entendimiento.

La herramienta Flex es una herramienta y un auxiliar para generadores de analizadores léxicos. Su propósito no es más que ayudar a generar analizadores con la ayuda expresiones regulares.

También nos permitió la herramienta y auxiliar para generar un analizador léxico que nos ayudó a generar analizadores con la ayuda expresiones regulares.

Referencias

- Manual de Flex Vern Paxson, Will Estes and John Millaway
- The C Programming, Prentice Hall, 1978, Brian Kernighan y Dennis Ritchie,