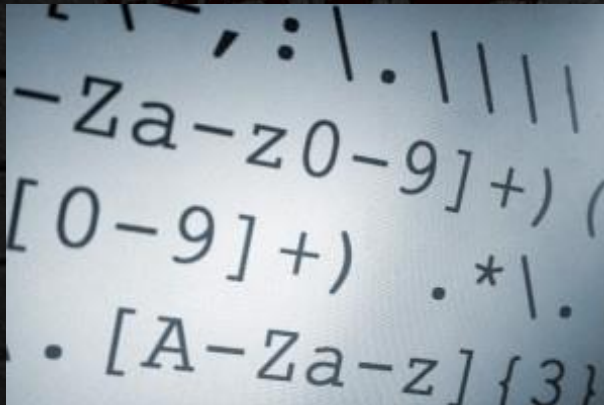




Generador de analizadores léxicos FLEX



Compiladores
Prof. Luis Enrique Hernández Olvera



Introducción a Flex

- Abrimos la terminal (Ctrl + Alt + T) y escribimos estos comandos:
 - `sudo apt-get update`
 - `sudo apt-get install flex`

Compiladores
Prof. Luis Enrique Hernández Olvera

2



Introducción a Flex

- Flex es una herramienta que permite generar **escáneres**: (programas que reconocen patrones léxicos en un texto).
- A partir de un conjunto de expresiones regulares, Flex busca concordancias en un fichero de entrada y ejecuta acciones asociadas a estas expresiones.

Compiladores
Prof. Luis Enrique Hernández Olvera

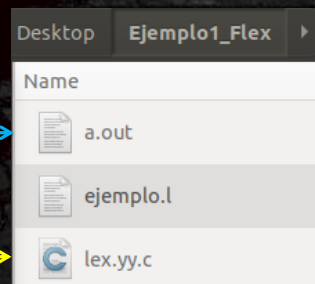
3



Introducción a Flex

- Con los ficheros de entrada (extensión .l), flex genera como salida un fichero fuente en C, 'lex.yy.c', que define una rutina 'yylex()'. Este fichero se compila y se enlaza con la librería '-lfl' para producir un ejecutable.
- \$ flex prueba.l
- \$ gcc lex.yy.c -lfl

```
olvera@olvera-N551JW: ~/Desktop/Ejemplo1_Flex
Terminal Help
W:~/Desktop/Ejemplo1_Flex$ flex ejemplo.l
W:~/Desktop/Ejemplo1_Flex$ gcc lex.yy.c -lfl
W:~/Desktop/Ejemplo1_Flex$ ./a.out
```





Introducción a Flex

- Cuando se arranca el fichero ejecutable, este analiza su entrada en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el código C correspondiente.



Estructura del archivo Flex

- El fichero de entrada de flex está compuesto de tres secciones, separadas por una línea donde aparece únicamente un `%%`.

definiciones

%%

reglas

%%

código de usuario



sección de **definiciones**

La sección de **definiciones** contiene declaraciones de definiciones de **nombres** sencillas para simplificar la especificación del escáner. Las definiciones de nombre tienen la forma:

nombre definición



sección de **definiciones**

El "nombre" es una palabra que comienza con una letra o un guion bajo seguido por cero o más letras, dígitos, ``_``, o ``-``.

La definición se considera que comienza en el primer carácter que no sea un espacio en blanco siguiendo al nombre y continuando hasta el final de la línea. Posteriormente se puede hacer referencia a la definición utilizando "{nombre}", que se expandirá a "(definición)".



sección de definiciones

Ejemplo:

DIGITO [0-9]
LETRAS [A-Za-z]+

define "DIGITO" como una expresión regular que empareja un dígito sencillo, y "LETRAS" como una expresión regular que empareja una o mas letras ya sean mayúsculas o minúsculas.



Algunos ejemplos de expresiones en Flex

- Dígito => 0|1|2|3|4|5|6|7|8|9
- Dígito => [0-9]
- Número entero: [0-9]+
- Número entero: {Dígito}+
- Número real sin signo: {Dígito}+"."{Dígito}*
- Identificador => [a-zA-Z][_a-zA-Z]*
- Palabras que empiezan por "f o F": [fF][a-zA-Z]*
- Líneas que NO empiezan por "f o F": ^[^fF].*
- Literal "abc" exacto: abc
- Palabra "and": [aA][nN][dD]



sección de *reglas*

La sección de *reglas* en la entrada de flex contiene una serie de reglas de la forma:

patrón acción

Donde:

- **patrón:** expresión regular.
- **acción:** código C con las acciones a ejecutar cuando se encuentre concordancia del patrón con el texto de entrada



sección de *reglas*

El patrón debe estar sin sangrar y la acción debe comenzar en la misma línea.

Flex recorre la entrada hasta que encuentra una concordancia y ejecuta el código asociado. El texto que no concuerda con ningún patrón lo copia tal cual a la salida.



sección de *reglas*

Una regla mas elaborada se puede crear haciendo uso de las definiciones que previamente vimos.

Por ejemplo:

`{(DIGITO)}+ | {(DIGITO)}+ "." {(DIGITO)}+`

es idéntica a

`(([0-9])+ | ([0-9])+ "." ([0-9])+)`



sección de **código de usuario**

- La sección de código de usuario simplemente se copia a `lex.yy.c` literalmente.
- Esta sección se utiliza para rutinas de complemento que llaman al escáner o son llamadas por este.
- La presencia de esta sección es opcional; Si se omite, el segundo `%%` en el fichero de entrada se podría omitir también.



- En las secciones de definiciones y reglas, cualquier texto *sangrado* o encerrado entre ``%{'` y ``%}'` se copia íntegramente a la salida (sin los `%{'`'s). Los `%{'`'s deben aparecer sin sangrar en líneas ocupadas únicamente por estos.
- En la sección de definiciones (pero no en la sección de reglas), un comentario sin sangría (es decir, una línea comenzando con `"/*`") también se copia literalmente a la salida hasta el próximo `"*/"`.



- En la sección de reglas, cualquier texto o `%{'` sangrado que aparezca antes de la primera regla podría utilizarse para declarar variables que son locales a la rutina de análisis y (después de las declaraciones) al código que debe ejecutarse siempre que se entra a la rutina de análisis. Cualquier otro texto sangrado o `%{'` en la sección de reglas sigue copiándose a la salida.



Patrones

- Los patrones en la entrada se escriben utilizando un conjunto extendido de expresiones regulares y usando como alfabeto cualquier carácter ASCII. Cualquier símbolo excepto el espacio en blanco, tabulador, cambio de línea y los caracteres especiales se escriben tal cual en las expresiones regulares (patrones) de Flex. Los caracteres especiales son:

`" \ [^ - ? . * + | () $ / { } % < >`



Patrones

Algunos de los patrones de Flex son:

Exp.	Descripción
<code>x</code>	empareja el carácter <code>`x'</code>
<code>.</code>	cualquier carácter (byte) excepto una línea nueva
<code>[xyz]</code>	una "clase de caracteres"; en este caso, el patrón empareja una <code>`x'</code> , una <code>`y'</code> , o una <code>`z'</code>
<code>[abj-oz]</code>	una "clase de caracteres" con un rango; empareja una <code>`a'</code> , una <code>`b'</code> , cualquier letra desde la <code>`j'</code> hasta la <code>`o'</code> , o una <code>`z'</code>



Patrones

[^A-Z]	una "clase de caracteres negada", es decir, cualquier caracter menos los que aparecen en la clase. En este caso, cualquier caracter EXCEPTO una letra mayúscula.
[^A-Z\n]	cualquier caracter EXCEPTO una letra mayúscula o una línea nueva.
r*	cero o más r's, donde r es cualquier expresión regular.
r+	una o más r's
r?	cero o una r (es decir, "una r opcional").
r{2,5}	donde sea de dos a cinco r's.



Patrones

r{2, }	dos o más r's.
r{4}	exactamente 4 r's.
{nombre}	la expansión de la definición de "nombre".
"[xyz]\\"foo"	la cadena literal: [xyz]"foo
\x	si x es una `a', `b', `f', `n', `r', `t', o `v', entonces la interpretación ANSI-C de \x. En otro caso, un literal `x' (usado para indicar operadores tales como `*').
(r)	empareja una R; los paréntesis se utilizan para anular la precedencia.



Patrones

rs	la expresión regular r seguida por la expresión regular s ; se denomina "concatenación".
$r s$	bien una r o una s .
r/s	una r pero sólo si va seguida por una s .
$\wedge r$	una r , pero sólo al comienzo de una línea (es decir, justo al comienzo del análisis, o a la derecha después de que se haya analizado una línea nueva).
$r\$$	una r , pero sólo al final de una línea (es decir, justo antes de una línea nueva). Equivalente a " $r/\backslash n$ ".



Emparejamiento de la entrada

- Cuando el escáner generado está funcionando, este analiza su entrada buscando cadenas que concuerden con cualquiera de sus patrones. Si encuentra más de un emparejamiento, toma el que empareje el texto más largo. Si encuentra dos o más emparejamientos de la misma longitud, se escoge la regla listada en primer lugar en el fichero de entrada de Flex.



Emparejamiento de la entrada

- Una vez que se determina el emparejamiento, la *acción* correspondiente al patrón emparejado se ejecuta y luego la entrada restante se analiza para otro emparejamiento.
- Si no se encuentra un emparejamiento, entonces se ejecuta la **regla por defecto**: el siguiente carácter en la entrada se considera reconocido y se copia a la salida estándar.



Emparejamiento de la entrada

Por ejemplo:

%%

aa {printf("1");}

aab {printf("2");}

uv {printf("3");}

xu {printf("4");}

Con el texto de entrada Laabdgf xuv, daría como salida L2dgf 4v. El ejecutable copiará L, reconocerá aab (porque es más largo que aa) y realizará el printf("2"), copiará dgf, reconocerá xu (porque aunque tienen la misma longitud que uv, y uv está antes en el fuente, en el momento de reconocer la x flex no reconoce el posible conflicto, puesto que sólo xu puede emparejarse con algo que empieza por x y ejecutará el printf("4") y luego copiará la v que falta).



Ejemplo

```

olvera@olvera-N551JM:~/Desktop/Ejemplo_Flex$ flex ejemplo.l
olvera@olvera-N551JM:~/Desktop/Ejemplo_Flex$ gcc lex.yy.c -lfl
olvera@olvera-N551JM:~/Desktop/Ejemplo_Flex$ ./a.out
5.5.5.5
ip valida
Salto de línea
Salto de línea
+
Signo op
Salto de línea
()
Signo parenthesis
Signo parenthesis
Salto de línea
255.255.255.256
ip valida
Numero entero 6
Salto de línea
Mod(5,3)
Modulo
Signo parenthesis
Numero entero 5
coma
Numero entero 3
Signo parenthesis
Salto de línea

```


Entradas

```

1 dir [0-9][1-9][0-9][1[0-9][0-9]|2[0-4][0-9]|25[0-5]
2 NUM [0-9]+
3 %%
4 {dir}."{dir}"."{dir}"."{dir} {
5     printf("ip valida\n");
6 }
7
8 {NUM} {
9     printf("Numero entero %s\n",yytext);
10 }
11
12 "+"|"*" {
13     printf("Signo op\n");
14 }
15
16 "("|")" {
17     printf("Signo parenthesis\n");
18 }
19
20 "," {
21     printf("coma\n");
22 }
23
24 "[Mm][0o][dD] {
25     printf("Modulo\n");
26 }
27
28 "\n" {
29     printf("Salto de linea\n");
30 }
31
32

```

Compiladores
Prof. Luis Enrique Hernández Olvera



Práctica 3

Diseñar las expresiones regulares en Flex que reconozcan:

- Nombres de variables.
- Números enteros con y sin signo.
- Números en forma decimal con y sin signo.
- Potencia (con signo o sin él) de números reales (con signo o sin él) .
- Operaciones matemáticas: +, -, *, /, MOD()

Compiladores
Prof. Luis Enrique Hernández Olvera