Introducción a Flex

Lenguajes de Programación y Procesadores de Lenguajes
Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Curso 2019-20

Índice

1. Introducción

Flex es un generador automático de analizadores léxicos. Tal y como muestra la Figura 1, Flex recibe como entrada un fichero de texto (normalmente con la extensión `.l`) con la especificación léxica y, a partir de este fichero, genera el código C correspondiente al analizador léxico.

Flex genera el analizador en un fichero fuente C que debe compilarse para producir un ejecutable. Cuando éste se ejecuta, analiza una cadena de entrada (desde la entrada estándar o un fichero) detectando posibles errores léxicos en la entrada.

La herramienta Flex se utiliza del siguiente modo:

1. Se ejecuta la herramienta pasándole como argumento el nombre del fichero (normalmente con extensión `.l`) donde se encuentra la especificación léxica (por ejemplo `alex.l`). Si la especificación es correcta, Flex genera el fichero (por defecto `lex.yy.c`) que constituye la implementación del analizador léxico.

   ```
   > flex alex.l
   ```

   En caso de querer cambiar el nombre por defecto del fichero que se genera se puede emplear la opción `-o`, tal y como se muestra a continuación:

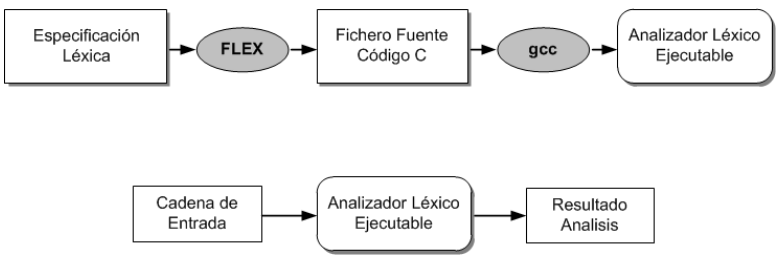

Figura 1: Uso de Flex

```
> flex -omianalizador.c alex.l
```

2. Se compila el fichero C generado, incluyendo la biblioteca de Flex `-lfl`.

```
> gcc -omianalizador mianalizador.c -lfl
```

3. Con la compilación anterior se obtiene el fichero ejecutable del analizador léxico.

   Para automatizar todos los pasos anteriores se puede crear un fichero *Makefile* y usar la orden *make*.

   Una vez creado el fichero ejecutable del analizador léxico, al ejecutarlo, por defecto se analizará la cadena que se lea de la entrada estándar. Si se desea leer la entrada desde un fichero, se indicará como parámetro el nombre del fichero en la orden de ejecución del analizador léxico:

```
> mianalizador ejemplo.c
```

2. Descripción de un analizador léxico mediante Flex

La especificación del analizador léxico en Flex se realiza a partir de parejas de patrones y acciones en código C denominadas *reglas*. El analizador léxico generado analizará la cadena de entrada en busca de cadenas de caracteres que casen con los patrones asociados a cada regla. Cuando encuentre alguna, ejecutará la acción en código C asociada a la regla correspondiente. Esta especificación se escribe en un fichero de texto en lenguaje Flex que consta de tres secciones separadas por una línea con los caracteres `%%`:

1. **Definiciones**: En esta sección se declara el código C necesario para el buen funcionamiento del código C que se va a utilizar o generar. También se pueden definir expresiones regulares, a las que se les dará un nombre para poder usarse posteriormente en la sección de reglas.
2. **Reglas**: En esta sección se definen las reglas que constituyen la especificación léxica.
3. **Funciones de usuario**: En está sección se definen funciones C que se incluirán en el fichero C generado. Estas funciones pueden ser llamadas desde las acciones de las reglas.

A continuación se describe con más detalle cada una de estas secciones.

2.1. Sección de definiciones

Esta primera sección está dividida en dos subsecciones. En la primera, *subsección preámbulo C*, se puede poner cualquier código escrito en C que se desee que aparezca al principio del código generado por Flex. En la segunda, *subsección de definiciones Flex*, aparecen opciones de configuración de Flex así como definiciones regulares (nombres asociados a expresiones regulares) que posteriormente se podrán usar en la sección de reglas para formar patrones más complejos del analizador léxico.

Veamos un ejemplo:

```
/* Subseccion preambulo */
%{
  #include <stdio.h>
  extern FILE *yyin;
%}

/* Subseccion de definiciones */

%option yylineno
```

```
delimitador     [ \t\n]+
digito          [0-9]
entero          {digito}+
```

En el ejemplo anterior se puede observar que la primera subsección, incluida entre delimitadores `%{` y `%}`, introduce el código de cabecera en C que se necesita: la biblioteca `stdio.h`. También se declara como externa la variable global `FILE *yyin` que se usará posteriormente para indicar a Flex que el analizador debe leer la cadena de entrada desde un fichero (por defecto el analizador léxico generado por Flex espera leer de la entrada estándar). También es posible incluir en esta misma sección cabeceras de ficheros C creados por el usuario. De esta forma pueden ponerse todas las declaraciones que necesite el código del usuario en un fichero, llamado por ejemplo "header.h", y añadir una orden `#include "header.h"` en esta sección.

La segunda subsección contiene la opción `yylineno` (la cual está precedida de la clausula `%option`), que ordena a Flex generar un analizador que mantenga el número de la línea actual en la variable global `yylineno` (útil para mostrar junto a los mensajes de error la línea en la que se ha producido).

A continuación se pueden incluir definiciones de expresiones regulares, cada una con un nombre asociado. Posteriormente, en la sección de reglas, se podrán usar los nombres (como si fuesen macros de Flex) para definir patrones del analizador léxico más complejos. Veamos un ejemplo de definición regular:

```
delimitador     [ \t\n]+
```

En primer lugar aparece el *nombre* de la definición regular `delimitador` y después la *definición* en sí. La expresión regular empieza en el primer carácter que siga al nombre y no sea un espacio en blanco y llega hasta el final de la línea. En este caso la expresión regular llamada `delimitador` casará con cualquier cadena formada por uno o más espacios en blanco, tabuladores o fines de linea.

A continuación se muestra otro ejemplo de expresión regular. En este caso la primera expresión regular (`digito`) casará con cualquier dígito, mientras que la segunda (`entero`) casará con secuencia de uno o más dígitos. Puede observarse como al usar una definición regular, se pone su nombre entre llaves.

```
digito          [0-9]
entero          {digito}+
```

En el Anexo A se puede encontrar una tabla con más operadores para construir expresiones regulares.

2.2. Sección de reglas

En la sección de reglas del analizador léxico es donde cada regla se define como el par *"patrón - acción"*. El *patrón* representa la definición de los símbolos léxicos que el analizador léxico debe reconocer y la *acción* las instrucciones escritas en C que se ejecutarán al reconocer un símbolo léxico que case con el patrón. En el Anexo A se puede encontrar una tabla con alguno de los operadores más usados para construir expresiones regulares en Flex.

Veamos un pequeño ejemplo:

```
{delimitador}   { }
"+"             { printf("operador:    %s\n", yytext); }
"-"             { printf("operador:    %s\n", yytext); }
"*"             { printf("operador:    %s\n", yytext); }
"/"             { printf("operador:    %s\n", yytext); }
"("             { printf("delimitador: %s\n", yytext); }
")"             { printf("delimitador: %s\n", yytext); }
{entero}        { printf("constante:   %s\n", yytext); }
.               { yyerror("Caracter desconocido");     }
```

La primera regla de este ejemplo define un delimitador usando la expresión regular `delimitador` definida de la sección anterior (para lo cual se indica su nombre entre llaves). La segunda regla define un nuevo patrón que casará con el operador de suma `+` , mientras que la tercera hace lo propio para el operador diferencia `-`. El patrón de la última regla está formado únicamente por un punto. Con esta regla se detecta la aparición de cualquier carácter que no case con ninguno de los patrones definidos anteriormente.

Las *acciones* asociadas a estos patrones se escriben en el lenguaje C y entre llaves (como bloques de instrucciones). En el ejemplo anterior podemos observar que, cuando se reconoce un símbolo léxico, se imprime por pantalla el tipo de símbolo reconocido (operador, delimitador, constante,...) y su lexema. Como se puede ver, el lexema leído por el analizador léxico se puede mostrar fácilmente por pantalla usando la variable global de Flex `yytex`, en la cual siempre podremos encontrar el lexema reconocido por cada regla.

Finalmente, la última regla permite que si se detecta un carácter que no case con ninguno de los patrones anteriores, se muestre el mensaje `\"Caracter desconocido\"`.

Nota sobre conflictos léxicos:

Si una secuencia caracteres de la entrada casa con más de un patrón, el analizador generado por Flex escogerá la secuencia de caracteres más larga. En el caso de que dos o más secuencias de caracteres tengan la misma longitud, se seleccionará la regla que aparezca antes en el fichero Flex.

2.3. Sección de funciones de usuario

En esta sección opcional se pueden incluir funciones C escritas por el usuario. Hay que tener en cuenta que, si se define alguna función que se use en la sección de reglas, habrá que incluir su cabecera en el preámbulo C de la sección de definiciones. Por otro lado Flex genera el analizador léxico en código C, y todo programa en C debe tener al menos una función `main`. Flex genera una función `yylex()` que se encarga de iniciar el análisis léxico que puede ser llamada desde la función `main`.

A continuación vemos un ejemplo de función `main` que redirecciona la entrada de Flex para que lea desde el fichero pasado como argumento (`argv[1]`) y llama a la función `yylex()` para que comience el análisis léxico.

```
int main(int argc, char **argv) {
   if (argc==2)
   if ((yyin = fopen (argv[1], "r")) == NULL)
      fprintf (stderr, "Fichero no valido <%s>\n", argv[1]);
   else yylex();
   else fprintf (stderr, "Uso: %s <fichero>\n",argv[0]);

   return 0 ;
}
```

Por supuesto también es posible escribir la función `main` en un fichero distinto del que contiene la especificación para Flex. En este caso, será necesario añadir el nombre del fichero C que contiene la función `main` en la orden de llamada al compilador que compile todo el analizador léxico generado por Flex.

2.4. Analizador léxico. Ejemplo

En código fuente del analizador léxico generado por Flex se ejecuta llamando a la función `yylex()`. Por defecto el analizador leerá la cadena a analizar desde la entrada estándar, pero este comportamiento se puede modificar asignando a la variable `FILE*yyin` el descriptor del fichero a analizar. De igual forma se puede cambiar la salida estándar dando valor a la variable de Flex `FILE *yyout`.

Flex tiene definidas otras variables que pueden resultar útiles en las acciones que se asocien a los patrones. Por ejemplo, la variable `char* yytext` contiene el lexema reconocido, mientras que la variable `int yyleng` contiene la longitud de dicho lexema.

Se muestra a continuación el código completo de un analizador léxico que reconoce algunos símbolos empleados en expresiones matemáticas.

```
/*************************************************************************/
/**  Ejemplo de FLEX: S E M - 1            2018-2019 <jbenedi@dsic.upv.es> **/
/**  V. 19                                                              **/
/*************************************************************************/
%{
#include <stdio.h>
#include <string.h>
#include "header.h"

#define  retornar(x) {if (verbosidad) ECHO;}

%}
/*-----------------------------------------------------------------------
Dado que las funciones "input()" y "unput(c)" son costosas y no las
utilizaremos, evitaremos que se generen automaticamente sus rutinas
correspondientes desactivandolas mediante las siguientes opciones:
-----------------------------------------------------------------------*/
%option noinput
%option nounput
/*-----------------------------------------------------------------------
Para mantener el numero de linea actual en la variable global yylineno
-----------------------------------------------------------------------*/
%option yylineno

delimitador     [ \t\n]+
digito          [0-9]
entero          {digito}+

%%

{delimitador}   {if (verbosidad) ECHO ; }
"+"   { retornar (MAS_) ; }
"-"             { retornar (MENOS_) ; }
"*"             { retornar (POR_) ; }
"/"             { retornar (DIV_) ; }
"("             { retornar (OPAR_) ; }
")"             { retornar (DIV_) ; }
{entero}        { retornar (CTE_) ; }
. { yyerror("Caracter desconocido") ;}

%%
int verbosidad = FALSE;

/*************************************************************************/
void yyerror(const char *msg){
/*  Tratamiento de errores.                                              */
fprintf(stderr, "\nError en la linea %d: %s\n", yylineno, msg);
}

/*************************************************************************/
int main(int argc, char **argv) {
```

```
int i, n=1 ;

for (i=1; i<argc; ++i)
if (strcmp(argv[i], "-v")==0) { verbosidad = TRUE; n++; }
if (argc == n+1)
if ((yyin = fopen (argv[n], "r")) == NULL)
fprintf (stderr, "El fichero '%s' no es valido\n", argv[n]) ;
else yylex ();
else fprintf (stderr, "Uso: cmc [-v] fichero\n");

return (0);
}

/**************************************************************************/
```

Para probar que el analizador generado funciona correctamente podemos compilarlo y ejecutarlo pasándole como entrada un fichero (en nuestro caso se llama ejem2) con el siguiente contenido:

```
3 - 5 * (12 + 5) + ((7 + 2) / 3) @
```

A continuación se muestra el resultado de todo este proceso:

```
> make
flex -oalex.c src/alex.l
gcc -Wall -I./include alex.c -lfl -o cmc
>./cmc -v ejem2
3 - 5 * (12 + 5) + ((7 + 2) / 3)
Error en la linea  1: Caracter desconocido
```

A. Expresiones regulares

Para definir expresiones regulares en Flex, tanto para la sección de definiciones como para la de reglas, se pueden emplear los operadores Flex que aparecen en la Tabla 1.

x	empareja el carácter 'x'
.	cualquier carácter (byte) excepto una línea nueva
[xyz]	una "clase de caracteres"; en este caso, el patrón empareja una 'x', una 'y', o una 'z'
[abj-oZ]	una "clase de caracteres con un rango; empareja una 'a', una 'b', cualquier letra desde la 'j' hasta la 'o', o una 'Z'
[^A-Z]	una "clase de caracteres negada", es decir, cualquier carácter menos los que aparecen en la clase. En este caso, cualquier carácter EXCEPTO una letra mayúscula
[^A-Z\n]	cualquier carácter 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
r{2,}	dos o más r's
r{4}	exactamente 4 r's
{nombre}	la expansión de la definición de "nombre" (ver más abajo)
"[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 '*')
\0	un carácter NUL (código ASCII 0)
\123	el carácter con valor octal 123
\x2a	el carácter con valor hexadecimal 2a
(r)	empareja una r; los paréntesis se utilizan para anular la precedencia (ver más abajo)
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. El texto emparejado por s se incluye cuando se determina si esta regla es el "emparejamiento más largo", pero se devuelve entonces a la entrada antes que se ejecute la acción. Así que la acción sólo ve el texto emparejado por r. Este tipo de patrones se llama "de contexto posterior". (Hay algunas combinaciones de r/s que Flex no puede emparejar correctamente).
^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).
<<EOF>>	un fin-de-fichero

Tabla 1: Expresiones regulares en Flex.