# Lenguajes de Programación y Procesadores de Lenguajes 2016/17
# INTRODUCCIÓN A FLEX

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

V.16

## 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.

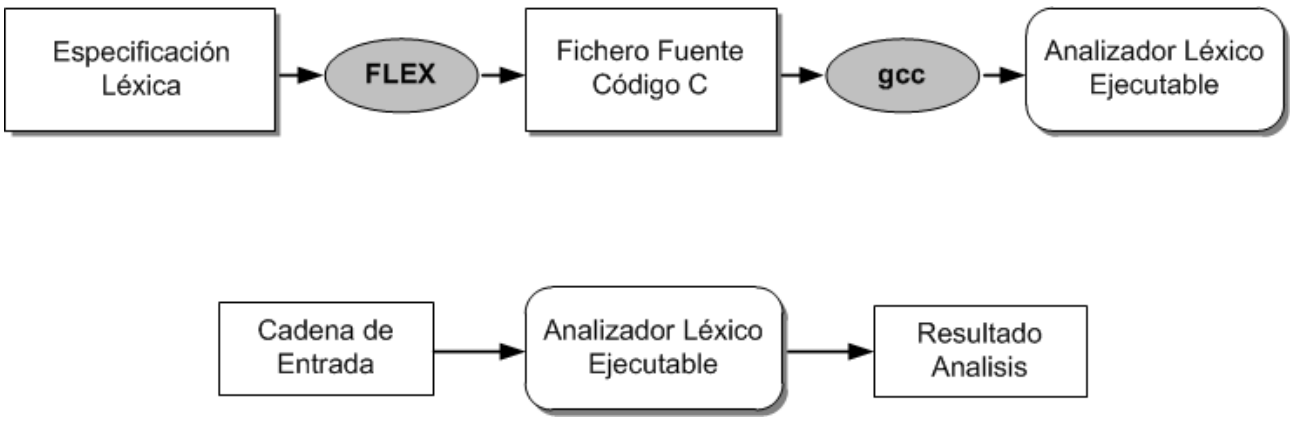


Figura 1: Uso de Flex

```
> flex alex.l
```

En caso de querer cambiar el nombre del fichero por defecto se puede indicar en línea de comandos tal y como se muestra a continuación:

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

2. Se compila el fichero C generado y se enlaza con la biblioteca de Flex `-lfl`.

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

3. Con la compilación anterior, se obtiene el fichero ejecutable del analizador léxico. Se puede ejecuvar el analizador léxico para analizar la entrada estándar (sin argumentos) o el texto de un fichero (pasado por argumento).

```
> 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*. Una vez generado el analizador léxico, éste analizará la cadena de entrada en busca de casos que casen con los patrones asociados a cada regla. Si encuentra algún caso ejecutará la acción en código C asociada a la regla correspondiente. Esta especificación se escribe en un fichero fuente 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, así como expresiones regulares que pueden utilizarse en la definición de los patrones de las 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 mas 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 C */
%{
#include <stdio.h>
extern FILE *yyin;
%}

/* Subseccion de definiciones */
%option yylineno

delimitador [ \t]+
```

```
letra [a-zA-Z]
digito [0-9]
```

En el ejemplo anterior, ses 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 la variable global `FILE *yyin` con el fin de indicarle a Flex que el analizador ddebe leer la cadena de entrada desde un fichero (por defecto el analizador léxico generado por Flex espera leer de la entrada estándar). Esto se indica en la sección de funciones de usuario tal y como veremos más adelante.

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 aparecen las definiciones de expresiones regulares, lcada una con un nombre asociado. De esta manera posteriormente se podrán usar los nombres (como si fuesen macros de Flex) para definir patrones del analizador léxico más complejos. Veamos un ejemplo:

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

Se puede observar que en primer lugar aparece el *nombre* 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 o tabuladores.

A continuación se muestra otro ejemplo de expresión regular. En este caso la primera expresión regular casará con cualquier letra en mayúscula o minúscula, mientras que la segunda casará con cualquier dígito.

```
letra [a-zA-Z]  digito [0-9]
```

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

## 2.2. Sección de reglas

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

Veamos un pequeño ejemplo:

```
{delimitador} {ECHO;}
"\n"          {ECHO; printf("%3d-",yylineno);}
{letra}       {ECHO; printf("%3d-",yylineno);}
{digito}      {ECHO; printf("%3d-",yylineno);}
if            {printf("P. Reservada:%s",yytex);}
for           {printf("P. Reservada:%s",yytex);}
.             {printf("Caracter no Valido");}
```

Los patrones asociados a cada una de estas reglas son los siguientes: la primera regla define un delimitador usando la expresión regular 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 retorno de carro `"\n"`. La tercera regla casará con la expresión regular definida para letras mayúsculas. A continuación se muestran dos reglas que definen las palabras reservadas `if` y `for`. Finalmente, se puede observar que el patrón de la última regla está formado únicamente por un punto. Con esta regla, que es la última que aparece en el fichero, se detecta la aparición de cualquier carácter que no case con ninguno de los patrones definidos anteriormente.

Las *acciones* asociadas a estos patrones se presentan escritas entre llaves (como bloques de instrucciones) y son las siguientes: para {`delimitador`} únicamente se debe imprimir por la salida estándar el lexema (mediante la instrucción de Flex `ECHO`). Para el separador `"\n"` y la expresión regular {`letra`} su *acción* indica que se muestre por salida estándar el retorno de carro o la letra (el lexema leído) seguido del número de línea en curso (formateado a tres dígitos).

Como se puede ver, el lexema leído por el analizador léxico se puede mostrar fácilmente por pantalla mediante la instrucción `ECHO`. Sin embargo, si se desea realizar cualquier otro tipo de operación con este lexema, la instrucción `ECHO` no es suficiente. Para ello, Flex proporciona la variable global `yytex`, en la cual se guarda el lexema leído por cada regla. Así pues, en la acción de cada regla se puede acceder al lexema leído a través de la variable `yytex` utilizando código C. Esto se puede ver en las acciones de las dos reglas que reconocen la palabras reservadas `if` y `for` que se encargan de visualizar el lexema leído precedido del texto `"P. Reservada"`. En esta caso sin embargo, en lugar de utilizar la instrucción `ECHO`, se ha visualizado el contenido de la variable `yytex` a través de la instrucción `printf` de C.

Finalmente, si se detecta cualquier otro carácter, se mostrará el mensaje `"Caracter no Valido"`.

**Nota sobre conflictos léxicos:**

Si una secuencia de 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 podemos incluir en la función `main`.

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

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

     if ((yyin = fopen (argv[1], "r")) == NULL)
            fprintf (stderr,
               "Fichero no valido "%s", argv[1]);
      yylex();

}
```

## 2.4. Ejemplo completo: Expresiones matemáticas

A continuación se muestra el código completo del ejemplo estudiado a lo largo de las secciones anteriores. Se han modificado las acciones asociadas a las reglas de forma que se han sustituido las llamadas a la función estándar de C *printf* por llamadas a una función de usuario *visualizar*.

```
/* Subseccion preambulo C */
%{

#include <stdio.h>
extern FILE *yyin;
void visualizar(int opcion, char* texto);
%}
```

```
/* Subseccion de definiciones */

%option yylineno

delimitador [ \t]+
letra [a-zA-Z]
digito [0-9]

%%

{delimitador}     {ECHO;}
"\n"                        {visualizar(0,"");}
if                          {visualizar(1,"P. Reservada: ");}
for                         {visualizar(1,"P. Reservada: ");}
"+"                         {visualizar(1,"Op. suma: ");}
"*"                         {visualizar(1,"Op. mult: ");}
{digito}+                   {visualizar(1,"Cte.: ");}
{letra}({letra}|{digito})*  {visualizar(1,"Identificador: ");}
.                           {visualizar(1,"Caracter no Valido:");}

%%

void visualizar(int opcion, char* texto)
{
    switch(opcion){
        case 0: printf("%3d- Nueva Linea\n",yylineno);
                break;
        case 1: printf("%3d- \%s \%s\n",yylineno,texto, yytext);
                break;
    }
}

int main (int argc, char **argv)
{
    if ((yyin = fopen (argv[1], "r")) == NULL)
        fprintf (stderr, "Fichero no valido \%s", argv[1]);
    yylex();
}
```

Para probar que el analizador generado funciona correctamente se puede ejecutar pasándole un fichero con la siguiente cadena de texto:

```
if A temp
5+99*83
@
```

El resultado obtenido será el siguiente:

```
 1- P. Reservada: if
   1- Identificador: A
   1- Identificador: temp
 2- Nueva Linea
   2- Cte: 5
   2- Op. suma: +
```

```
  2- Cte: 99
  2- Op. mult.: *
  2- Cte: 83
3- Nueva Linea
  3- Caracter no Valido: @
4- Nueva Linea
```

# 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.