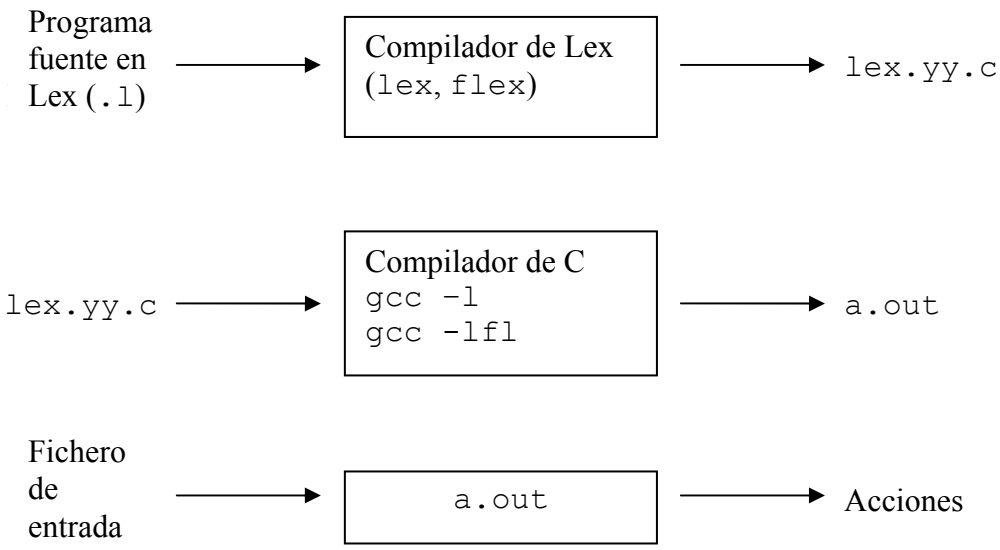
El generador de analizadores léxicos *lex*.

Teoría de Autómatas y lenguajes formales
Federico Simmross Wattenberg (fedesim@infor.uva.es)
Universidad de Valladolid

Una vez visto cómo las expresiones regulares pueden ser muy útiles a la hora de reconocer patrones en un fichero de texto, y de utilizarlas para desencadenar ciertas acciones en respuesta, vamos a estudiar una herramienta que permite crear programas autónomos que analizan léxicamente la entrada y la procesan a nuestro antojo: el generador de analizadores léxicos *lex*. Lex genera código fuente en C, a partir de una serie de especificaciones escritas en lenguaje Lex. El código C generado contiene una función llamada `yylex()`, que localiza cadenas en la entrada (lexemas) que se ajusten a uno de los patrones léxicos especificados en el código fuente Lex, realizando entonces las acciones asociadas a dicho patrón. `yylex()` puede llevar a cabo cualquier tipo de acciones ante un determinado patrón y, en particular, puede comportarse como un analizador léxico.

1. Funcionamiento de *lex*

Al contrario que `sed` y `awk`, lex no es un analizador sino un generador de analizadores. Esto permite incluir de manera cómoda un analizador a medida en cualquier programa:



2. El lenguaje Lex

2.1. Esquema general

Un programa fuente de Lex tiene el siguiente aspecto:

```
<sección de definiciones>
%%
<sección de reglas>
%%
<sección de rutinas>
```

- De estas tres secciones, sólo la segunda es obligatoria, y cualquiera de ellas puede estar vacía. Esto quiere decir que el mínimo programa en lex es:

  %%

- La sección de declaraciones incluye declaraciones de variables, constantes y *definiciones regulares*, que constituyen una manera cómoda de utilizar expresiones regulares largas en la sección de reglas; por ejemplo:

  ```
  letra          [A-Za-z]
  ```

- La sección de reglas especifica los patrones a reconocer y las acciones asociadas a éstos, de forma similar a la que utiliza `awk`:

  ```
  patrón         {acciones en C}
  ```

- La sección de rutinas permite definir funciones auxiliares en C, incluida la función `main()`. Por defecto, lex proporciona un `main()` que simplemente llama a la función `yylex()`.

El comportamiento del programa generado (por defecto a.out, si no se le indica otra cosa al compilador de C) es el siguiente:

- Imprimir en la salida estándar los lexemas que no se adapten a ningún patrón.
- Realizar la acción indicada para los lexemas que se ajustan a un patrón.

Por lo tanto, si compilamos y ejecutamos el programa mínimo en lex, que consta únicamente de una sección de reglas vacía (%%), veremos que se limita a copiar literalmente la entrada estándar en la salida estándar.

2.2. Sección de reglas

A primera vista, una regla de lex tiene el mismo aspecto que una sentencia patrón-acción de awk. Su estructura es:

```
patrón_regular  acción_en_C
```

- El patrón regular debe estar situado en la primera posición de la línea; es decir, que no puede haber espacios antes del patrón regular. Lex admite todas las expresiones regulares de `egrep`, con alguna extensión más (véase la página `man`).

- El patrón y la acción se separan por un blanco o un tabulador (o más).

- La acción puede ser una sola sentencia de C, o una sentencia compuesta, encerrada entre llaves {}.

- Cuando hay más de un patrón regular que puede adaptarse a la entrada, lex tomará la cadena más larga posible que pueda adaptarse a un patrón. Si aún así hay más de un patrón que se adapta a la entrada, lex ejecutará la acción de la regla que antes aparezca en el fuente lex.

- Lex almacena el lexema encontrado en un array de tipo char llamado `yytext`, y su longitud en la variable `yyleng`.

- La acción especial ECHO copia el lexema reconocido en la salida estándar. Es un sinónimo de `printf("%s",yytext);`

- La acción especial | (barra vertical) indica que para este patrón debe ejecutarse la acción correspondiente al patrón inmediatamente inferior.

- Todo programa lex incorpora automáticamente la siguiente línea como última regla:

  ```
  .|\n ECHO;
  ```

  equivalente a estas dos reglas separadas:

  ```
  .           |
  \n          ECHO;
  ```

  De manera que todos los lexemas que no encajan con ningún otro patrón, se copian en la salida estándar. Se puede evitar este comportamiento especificando reglas para `.` y para `\n`.

2.3. Código C en un programa lex

Muchas veces es útil poder incluir código en C dentro de un programa en lex, para apoyar el trabajo del analizador. Suele ser normal, por ejemplo, declarar una estructura de datos que contenga detalles sobre cada lexema encontrado o incluir nuestra propia función `main()`.

Puede insertarse código C en un fuente Lex en varias partes:

- En la sección de declaraciones, entre una línea `%{` y una línea `%}` (nótese que no es `}%`). Este código será externo y se situará antes de la función `yylex()` en el programa `lex.yy.c`.

- En la sección de declaraciones, cualquier línea que comience por un espacio en blanco se considerará código C y será también externo y anterior a `yylex()`.

- En la sección de reglas, y antes de que empiece la primera regla, toda línea que comience por un espacio en blanco se considerará código C que será interno a la función `yylex()`. Normalmente no es necesario incluir código C de esta manera.

- En la sección de reglas, las acciones en C de cada regla serán parte del código de `yylex()`.

- Toda la sección de rutinas es código C, externo y posterior a la función `yylex()`. Puede utilizarse para cualquier cosa, pero en particular, puede usarse para definir una función `main()` distinta a la que se genera por defecto.

3. Ejemplos

- Sustituir las palabras ‘el’ y ‘la’, ‘los’ y ‘las’ por la palabra ‘ARTICULO’.

```
%%
el|la|los|las       printf ("ARTICULO");
```

ó

```
%%
el      |
la      |
los     |
las     printf ("ARTICULO");
```

- Lo mismo, pero sin afectar a palabras como ‘lanza’:

```
%%
[Ee]l         |
[Ll][oa]s?    printf ("articulo");
[A-Za-z]+     ECHO;
```

- Pasar a mayúsculas los artículos determinados:

```
%%
[Ee]l         |
[Ll][ao]s?    { int i;
                for (i=0;i<yyleng;i++)
                     printf ("%c",toupper(yytext[i]));
              }
```

- Contar el número de artículos determinados:

```
%{
  /* Cuenta el número de artículos determinados */
  int  num=0;
%}

%%
[Ee]l             |
[Ll][oa]s?        num++ ;
[A-Za-z]+         ;
.|\n              ;

%%
main(){
   yylex();
   printf ("Número de artículos: %d\n", num);
}
```

4. Ejercicios

1- Compilar el programa mínimo lex (%%) y examinar el contenido del `lex.yy.c` generado.
2- Quitar los comentarios del fichero `findip.c`
3- Mostrar los comentarios del fichero `findip.c` (sólo los comentarios).
4- Quitar las palabras que consten de 3 letras o menos.
5- Construir un programa que imite al comando `wc` (contar los caracteres, palabras y líneas de un fichero).