

## *FLEX: Un generador de analizadores léxicos*

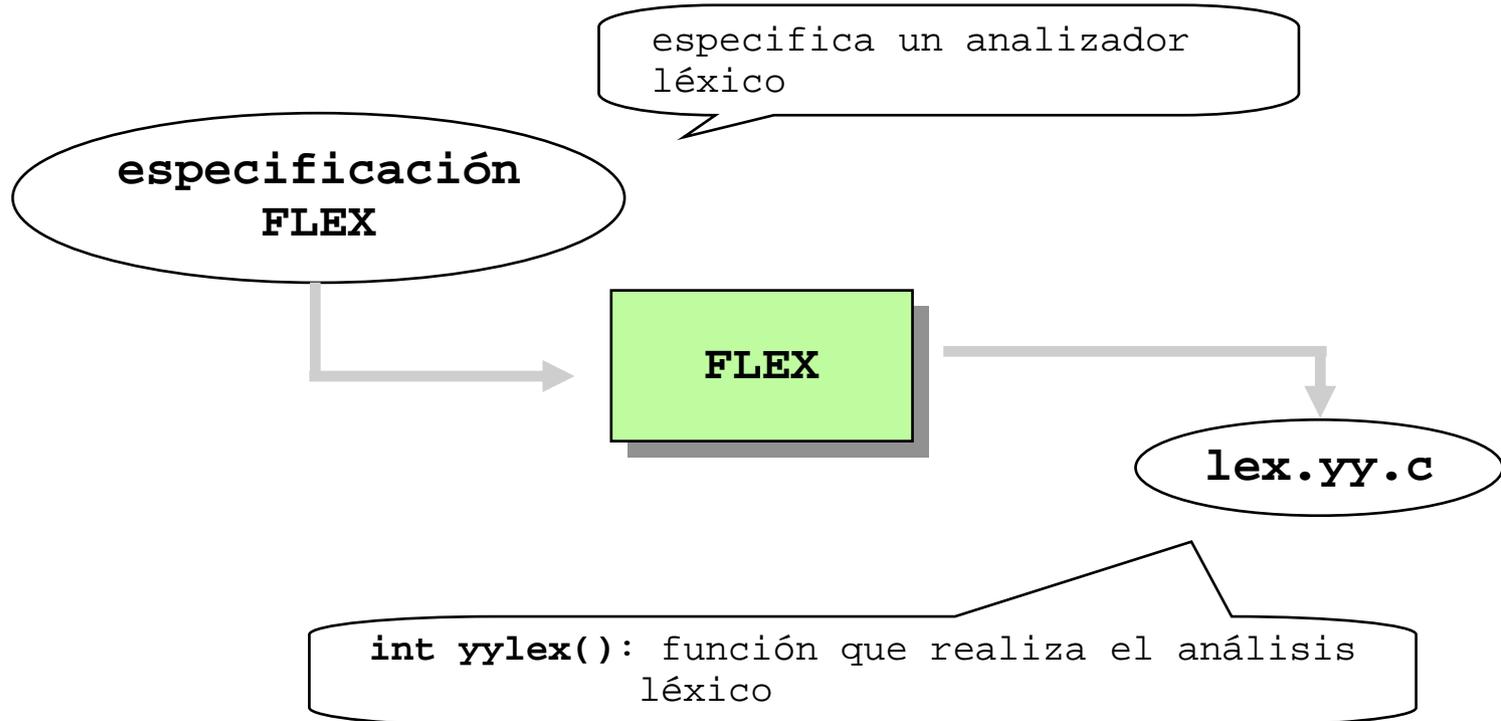
---

---

- Como hemos visto, el paso de una expresión regular a un AF se puede hacer de manera automatizada
- Existen varias herramientas que relizan dicho trabajo
- Generadores de analizadores léxicos
  - AT&T Lex (más común en UNIX)
  - MKS Lex (MS-Dos)
  - Flex
  - Abraxas Lex
  - Posix Lex
  - ScanGen
  - JLex
  - ...

## *FLEX: Un generador de analizadores léxicos*

- ¿Qué hace LEX?



## *FLEX: Un generador de analizadores léxicos*

- FLEX sigue el esquema:

|                                 |                                     |
|---------------------------------|-------------------------------------|
| <code>patrón<sub>1</sub></code> | <code>{ acción<sub>1</sub> }</code> |
| <code>patrón<sub>2</sub></code> | <code>{ acción<sub>2</sub> }</code> |
| <code>.....</code>              |                                     |
| <code>patrón<sub>k</sub></code> | <code>{ acción<sub>k</sub> }</code> |

donde:

- patrón: expresión regular
  - acción: fuente C con las acciones a realizar cuando el patrón concuerde con un lexema
- Funcionamiento:
    - FLEX recorre entrada estándar hasta que encuentra una concordancia
      - » un lexema correspondiente al lenguaje de algunas de las e.r. representadas por los patrones
    - entonces, ejecuta el código asociado (acción)
    - permite acceder a la información asociada al lexema (string, longitud del mismo, n<sup>o</sup> de línea en el fuente,...)

## *FLEX: Un generador de analizadores léxicos*

- Ejemplo: implementar en LEX, un analizador léxico para

```
menor → <
mayor → >
menorIgual → <=
mayorIgual → >=
igual → =
distinto → <>
letra → a|b|...|z|A|B|...|Z
digito → 0|1|...|9
identificador → letra (letra | digito)*
constEntera → digito digito *
```

## *FLEX: Un generador de analizadores léxicos*

- Comportamiento deseado:

v0<>27 segundos= 1000

analizador  
léxico

( IDENTIFICADOR , v0 )  
( DISTINTO , )  
( CONSTENTERA , 27 )  
( IDENTIFICADOR , segundos )  
( IGUAL , )  
( CONSTENTERA , 1000 )

## *FLEX: Un generador de analizadores léxicos*

- Estructura de un programa FLEX

```
sección de definiciones  
%%  
sección de reglas  
%%  
sección de rutinas del usuario
```

- **Sección de definiciones:**

- bloques literales
  - » se copian “tal cual” al fuente `lex.yy.c`
  - » entre ‘%{’ y ‘%}’
- definiciones regulares (“bautismo”)
- declaraciones propias para el manejo de tablas de LEX
- condiciones iniciales
  - » cambiar el funcionamiento del reconocedor

## FLEX: Un generador de analizadores léxicos

---

---

```
%{ /* fichero: expresiones.l */
#include ....
enum {MENOR=255,MENORIGUAL,MAYOR,
      MAYORIGUAL,IGUAL,DISTINTO,
      IDENTIFICADOR,CONSTENTERA};

int yylval;
      /* para atributo cte. entera*/
typedef int token;
%}
%%

      sección de reglas

%%

      sección de rutinas de usuario
```

- **Sección de reglas**

- Formato:

|                     |                         |
|---------------------|-------------------------|
| <code>patrón</code> | <code>{acciones}</code> |
|---------------------|-------------------------|

- dos tipos de líneas:

- » empiezan por blanco, '%{' ó '%}' fuente C

- » empiezan por otra cosa: línea patrón-código

- las líneas de fuente C se copian directamente en el fuente `lex.yy.c`

- al encontrar concordancia, ejecutará código asociado

- si no encuentra concordancia, ejecuta ECHO (copia el lexema en salida estándar)

## FLEX: Un generador de analizadores léxicos

```
%{
    sección de definiciones
}%
%%
(\t|\n|" ")+          { /* no hace nada */ };
[0-9]([0-9])*         { sscanf(yytext, "%d", &yy1val);
                        return(CONSTENTERA); }
"<"                  { return(MENOR); }
....
....
"<>"                 { return(DISTINTO); }
[a-zA-Z]([a-zA-Z]| [0-9])* { return(IDENTIFICADOR); }
.                      { ECHO; }
%%
    sección de rutinas de usuario
```

↑  
patrón

↑  
acción

- **Sección de rutinas de usuario**

- su contenido se copia literalmente en el fuente `lex.yy.c`
- contiene funciones C escritas por el usuario y necesarias para el analizador
  - » manejo de tabla de símbolos
  - » generación de salidas (cuando no es necesario generación de código)

## FLEX: Un generador de analizadores léxicos

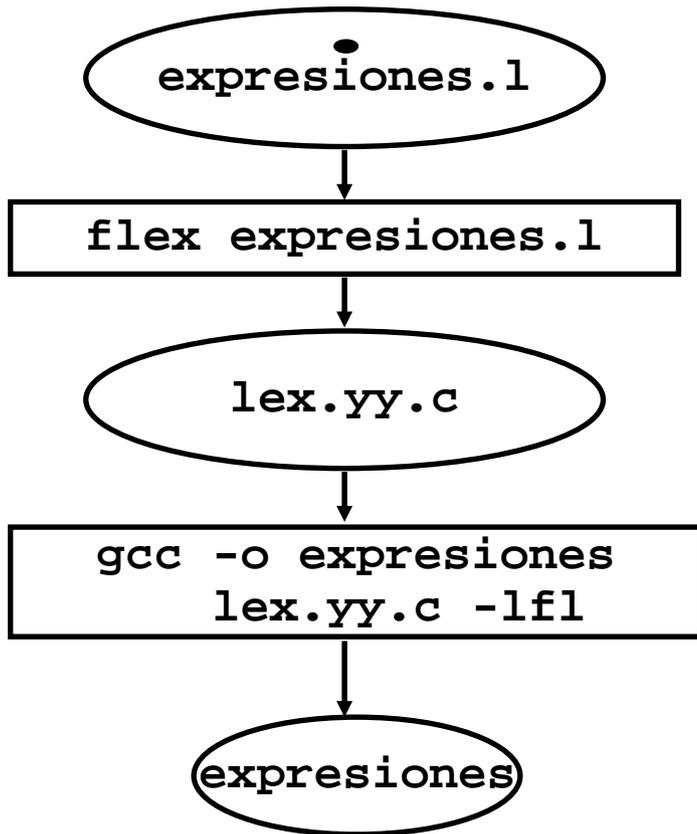
```
%{
    sección de definiciones
}%
%%
    sección de reglas
%%
/*-----
    Sólo para comprobación
-----
*/
void escribeInfo(token elToken)
{
    switch(elToken){
        case IDENTIFICADOR:
            fprintf(stdout,
                "ident.\t%s", yytext);
            break;
            .....
    }
}
```

```
/*-----
    Cuando EOF, yylex() dev. 0
-----*/
int main()
{
    token elToken;

    elToken=yylex();
    while(elToken){
        escribeInfo(elToken);
        elToken=yylex();
    }
}
```

## FLEX: Un generador de analizadores léxicos

- Ejemplo de uso de LEX:



sufijo ``.l'`  
para fuente FLEX

invocación a FLEX

fuentes con el  
analizador

`-lfl`: biblioteca  
necesaria

analizador

## *FLEX: Un generador de analizadores léxicos*

```
#=====
#  Fichero: Makefile
#  Tema:    genera un analizador léxico para
#           introducir Flex/Lex
#  Fecha:   Septiembre-03
#  Uso:     make
#=====
LEX=flex
CC=gcc

expresiones: lex.yy.o
    $(CC) -o expresiones lex.yy.o -lfl
    #-L/opt/flex/lib -lfl para Merlin
lex.yy.o: lex.yy.c
    $(CC) -c lex.yy.c

lex.yy.c: expresiones.l
    $(LEX) expresiones.l
```

## *FLEX: Un generador de analizadores léxicos*

- Ejemplo de invocación:

- el fichero 'entrada' es

```
v0<>27
segundos=      1000
```

- invocación:

```
expresiones < entrada
```

- genera la siguiente salida

```
identificador v0
distinto      <>
numero        27
identificador segundos
igual         =
numero        1000
```

## *FLEX: Un generador de analizadores léxicos*

- También se pueden dar nombres a expresiones regulares

```
%{      /* fichero: expresiones.l */
        .....
int yylval;      /* hab. los define Yacc */
typedef int token;
}%
separadores      [\n\t ]+
        .....
letra            [a-zA-Z]
digito          [0-9]
identificador   {letra}({letra}|{digito})*
constEntera    {digito}({digito})*
%%
                sección de reglas
%%
                sección de rutinas de usuario
```

## *FLEX: Un generador de analizadores léxicos*

```
%{  
    sección de definiciones  
%}  
%%  
{separadores}  { /* no hace nada */ };  
{constEntera} { sscanf(yytext, "%d", &yy1val);  
                return(CONSTENTERA); }  
  
.....  
{identificador} { return(IDENTIFICADOR); }  
.                { ECHO; }  
%%  
    sección de rutinas de usuario
```

**Nota:** el nombre de la ER en la parte de patrones debe ponerse entre '`{`' y '`}`' para distinguirlo del patrón literal

**`{separadores}<>separadores`**

## FLEX: Un generador de analizadores léxicos

- Extensiones de FLEX a e.r.

|   |  |
|---|--|
| <code>.</code>                                | cualquier carácter excepto " <code>\n</code> "   |
| <code>r*</code>                               | 0 ó más concat. de <code>r</code>  |
| <code>r+</code>                               | 1 ó más concat. de <code>r</code>  |
| <code>r?</code>                               | 0 ó 1 veces <code>r</code>   |
| <code>[c<sub>1</sub>...c<sub>n</sub>]</code>  | conj.caracteres {c <sub>1</sub> ...c <sub>n</sub> }  |
| <code>a-b</code>                              | caracteres {a,succ(a),...}   |
| <code>^r</code>                               | <code>r</code> debe concordar al principio de la línea   |
| <code>[^c<sub>1</sub>...c<sub>n</sub>]</code> | cualquier car.∉{c <sub>1</sub> ...c <sub>n</sub> }   |
| <code>r\$</code>                              | <code>r</code> debe concordar al terminar la línea   |
| <code>r{m,n}</code>                           | entre m y n concatenaciones de <code>r</code>  |
| <code>\</code>                                | para concordancia exacta de caracteres especiales: <code>\\</code> <code>\.</code> <code>\?</code> |
| <code>r1 r2</code>                            | lo habitual  |
| <code>"c<sub>1</sub>...c<sub>n</sub>"</code>  | literalmente c <sub>1</sub> ...c <sub>n</sub>  |
| <code>r1/r2</code>                            | reconoce <code>r1</code> sólo si va seguida de <code>r2</code> ( <b>¿e.r.?</b> )                   |

## FLEX: Un generador de analizadores léxicos

- ¿Qué pasa cuando hay ambigüedad?
- FLEX aplica dos reglas:

```
.....
letra          [a-zA-Z]
digito         [0-9]
identificador {letra}({letra}|{digito})*
%%
"if"           {return(IF);}
{identificador} {return(IDENT);}
%%
```

### Regla 1

Aplica el patrón que concuerde con el string más largo

### Regla 2

Si dos patrones representa el mismo string, aplica el que aparece antes en las declaraciones de LEX

## Un ejemplo: contar caracteres, líneas y palabras

```
#include <stdio.h>
int nCar=0, nPal=0, nLin=0;
%}
palabra  [^ \t\n]+
finLin   \n
%%
{palabra} {nCar+=yyleng;nPal++;}
{finLin}  {nCar++;nLin++;}
.         {nCar++;}
%%
int main(){
    yylex();
    fprintf(stdout,
        "car:%d pal:%d lin:%d\n",
        nCar, nPal, nLin
    );
}
```

Una versión  
"a las bravas"

## Un ejemplo: contar caracteres, líneas y palabras

```
#include <stdio.h>
enum {PALABRA=128,FINLINEA,
      OTROSEP};
int nCar=0, nPal=0, nLin=0;
%}
palabra  [^ \t\n]+
finLin   \n
%%
{palabra} {return PALABRA;}
{finLin}  {return FINLINEA;}
.         {return OTROSEP;}
%%
int main(){

```



```
int elToken;
elToken=yylex();

while(elToken){
  switch(elToken){
    case PALABRA: nPal++;nCar+=yyleng;
                 break;
    case FINLINEA:nCar++;nLin++;
                 break;
    case OTROSEP: nCar++;
                 }
  elToken=yylex();
}
fprintf(stdout,
        "car:%d pal:%d lin:%d\n",
        nCar, nPal, nLin
);
```

un poco más  
estructurada  
y elegante