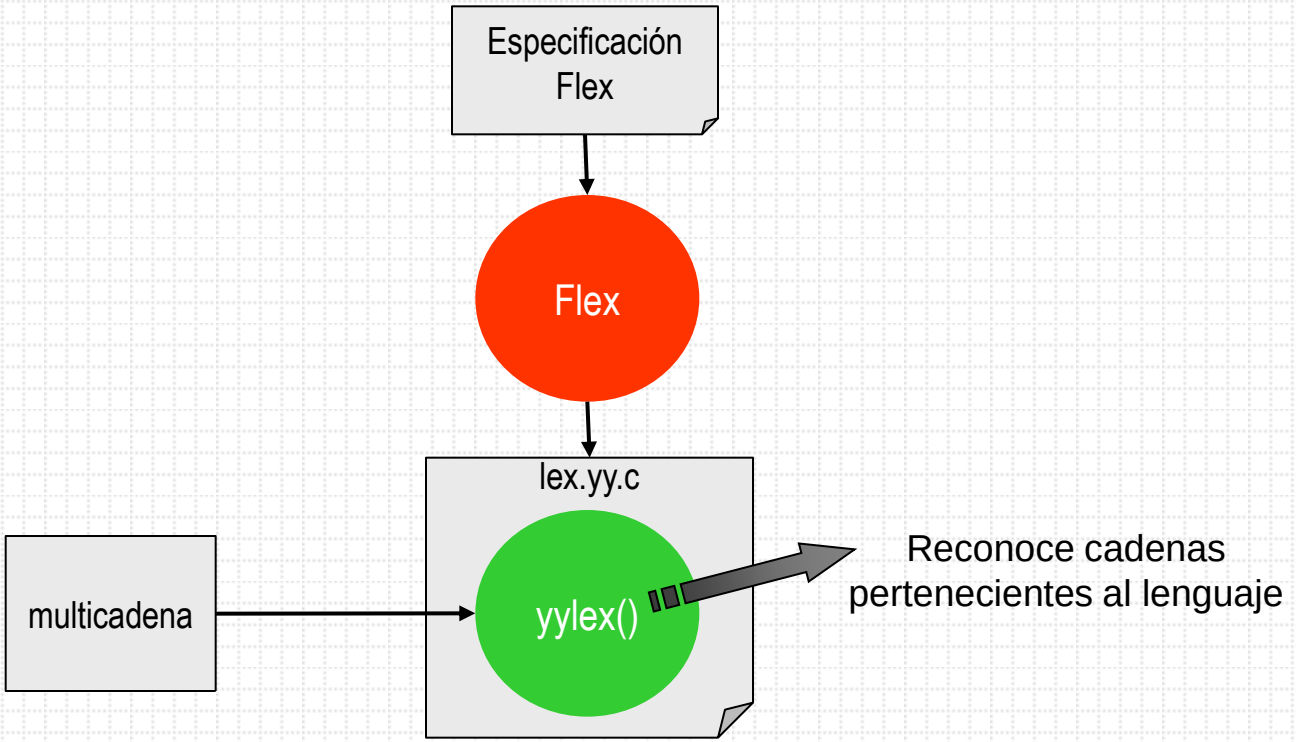
Índice (I)

Marina de la Cruz
Alfonso Ortega

Funcionamiento básico de Flex

- Flex es una herramienta que permite **generar automáticamente autómatas finitos que reconocen lenguajes regulares expresados mediante patrones Flex.**
- Los **patrones** de Flex **son extensiones de las expresiones regulares**.
- Flex recibe como **entrada un lenguaje regular expresado como un conjunto de patrones**, y genera la función C **yylex()** que es un autómata finito que **reconoce cadenas** pertenecientes a dicho lenguaje de entrada y opcionalmente **ejecuta alguna acción** cada vez que se reconoce una de las cadenas del lenguaje.
- A la entrada a la herramienta flex se le llama **especificación Flex**.



Descripción del primer ejemplo (I)

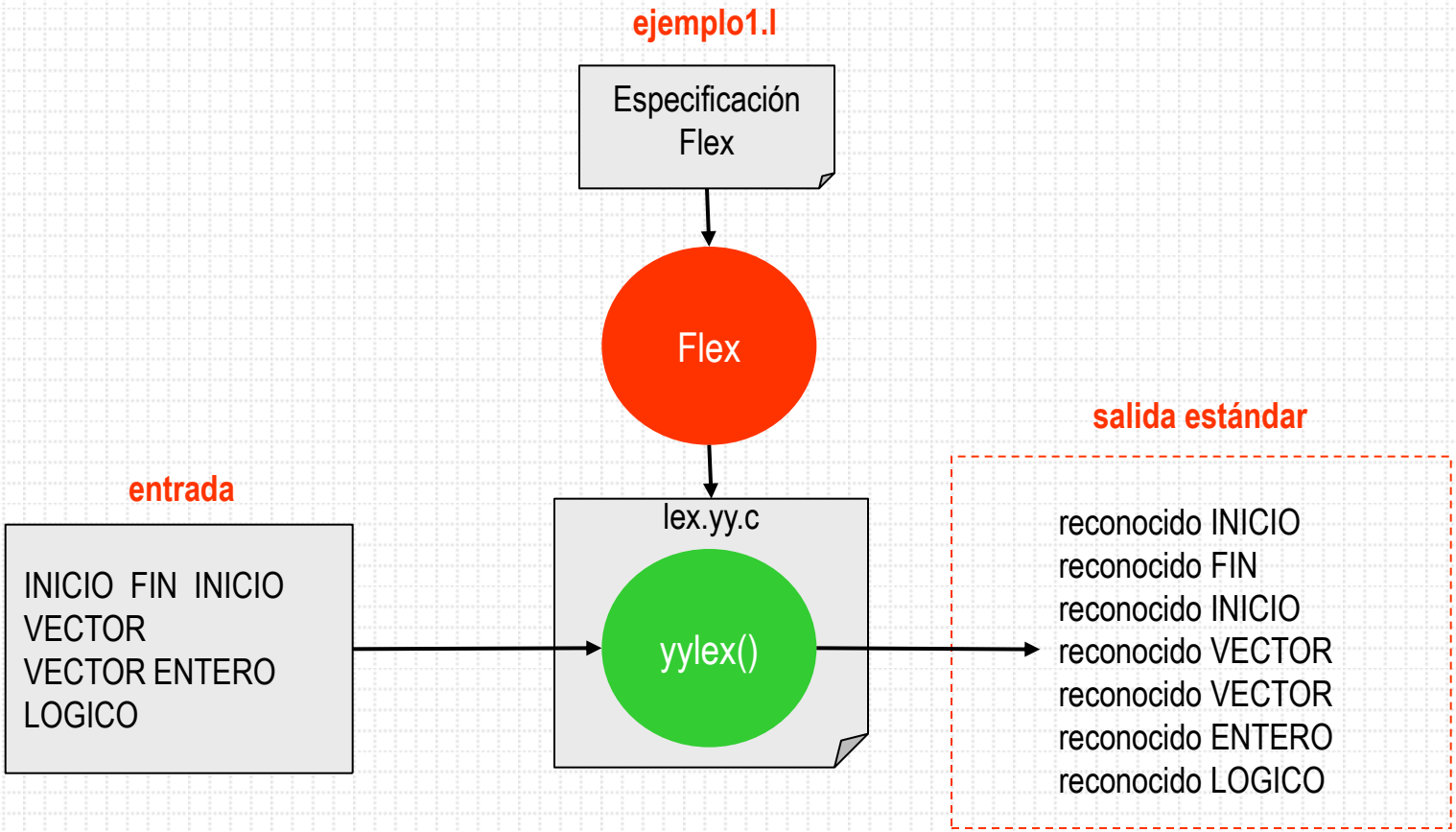
- Construir un autómata finito, utilizando la herramienta Flex, que cumpla las siguientes especificaciones:
  - reconoce las palabras “INICIO” , “FIN”, “VECTOR”, “ENTERO” y “LOGICO” .
  - Cada vez que el autómata reconoce una de las palabras anteriores, la acción que ejecuta es mostrar por la salida estándar un mensaje de aviso. Los mensajes indican qué palabra se ha identificado con el siguiente texto:

    “reconocido INICIO”
    “reconocido FIN”
    “reconocido VECTOR”
    “reconocido ENTERO”
    “reconocido LOGICO”

  - El fichero de especificación Flex se llamará **ejemplo1.l**.

Descripción del primer ejemplo (II)

- Una visión gráfica del primer ejemplo sería:



Estructura del fichero

- La estructura del fichero de especificación Flex se compone de tres secciones separadas por líneas que contienen el separador **%%**.

sección de definiciones

```
%{
    /* delimitadores de código C */
%}


%%
```

sección de reglas

```
%%
```

sección de funciones de usuario

La sección de definiciones (I)

- La sección de definiciones contiene la siguiente información
  - Código C encerrado entre líneas con los caracteres **%{** y **%}** que se copia literalmente en el fichero de salida **lex.yy.c** antes de la definición de la función **yylex()**. Habitualmente esta sección contiene declaraciones de variables y funciones que se utilizan posteriormente en la sección de reglas así como directivas **#include**.
  - Definiciones propias de Flex, que permiten asignar un nombre a una expresión regular o a una parte, y posteriormente utilizar ese nombre. Estas definiciones se verán con más detalle cuando se estudien los patrones de Flex.
  - Opciones de Flex similares a las opciones de la línea de comandos.
  - Cualquier línea que empiece con un espacio en blanco se copia literalmente en el fichero de salida **lex.yy.c**. Habitualmente se utiliza para incluir comentarios encerrados entre /* y */. (Esta funcionalidad depende de la versión de Flex)

La sección de definiciones (II)

- En la figura se muestra una primera aproximación del fichero de especificación Flex del primer ejemplo (ejemplo1.l) Únicamente se ha completado la sección de definiciones. A medida que se construya el ejemplo se completarán las demás secciones del fichero.

ejemplo1.l

```
%{
#include <stdio.h>          /* para utilizar printf en la
                               sección de reglas */
%}

%option noyywrap            /* para procesar sólo 1
                               fichero */


%%
```

- La directiva **#include <stdio.h>** se necesita porque en la sección de reglas se utiliza la función **printf** para mostrar en la salida estándar los mensajes de aviso de cadena reconocida. (Hay versiones de flex que hacen este include por defecto, la versión instalada en los laboratorios lo hace).
- El significado de la opción **noyywrap** se explica en el siguiente apartado.

La sección de definiciones (III)

- Significado de la opción **noyywrap**:
  - La función **yylex()** puede analizar varios ficheros, encadenando uno detrás de otro con el mecanismo que se describe a continuación.
  - Cuando **yylex()** encuentra un fin de fichero, llama a la función **yywrap()**. Si la función devuelve 0, el análisis continúa con otro fichero, y si devuelve 1, el análisis termina.
  - ¿Quién proporciona la función **yywrap()** ? En Linux, la librería de Flex proporciona una versión por defecto de **yywrap()** que devuelve 1. Hay que enlazar con esa librería con la opción -lf.
  - También se puede escribir una versión de la función **yywrap()** en la sección de código de usuario.

```
int yywrap() { return 1;}
```

  - La opción **noyywrap** evita que se invoque a la función **yywrap()** cuando se encuentre un fin de fichero, y se asuma que no hay más ficheros que analizar.
    Esta solución es más cómoda que tener que escribir la función o bien enlazar con alguna librería.

La sección de reglas (I)

- La sección de reglas contiene:
  - Los patrones que describen el lenguaje regular y las acciones escritas en C. Cada patrón se sitúa en una línea, seguido de uno o más espacios en blanco y a continuación el código C que se ejecuta cuando se encuentra dicho patrón en la entrada que se está analizando. El código C se cierra entre llaves {}.
  - Si una línea empieza por un espacio en blanco se considera código `C` y se copia literalmente en el fichero de salida. También se asume que es código `C` todo lo que se escriba entre `%{` y `%}`, y se copia literalmente en el fichero de salida. (Esta funcionalidad depende de la versión de Flex)

- ¿Cómo se comporta la rutina de análisis **yylex()** ?
  - Lee la entrada carácter a carácter y busca los patrones que se definen en la sección de reglas. Cada vez que se encuentra una cadena, se ejecuta el código C asociado con el patrón. Si no se identifica ninguna cadena, se ejecuta **la acción por defecto**: el carácter de la entrada se copia en la salida.
  - Si ninguna regla tiene una sentencia **return**, **yylex()** analiza la entrada hasta que encuentra el fin de la misma.
  - El código C asociado a cada patrón puede tener una sentencia **return**, que devuelve un valor al llamador de la función **yylex()** cuando se identifique el patrón en la entrada. La siguiente llamada a **yylex()** comienza a leer la entrada en el punto donde se quedó la última vez. Cuando se encuentra el fin de la entrada **yylex()** devuelve 0 y termina.

La sección de reglas (II)

- Sección de reglas del primer ejemplo:
  - Los patrones de Flex se estudiarán más adelante, y ya se verá que el patrón para representar una palabra es la misma palabra.
  - La sección de reglas del primer ejemplo contiene:
    - Una regla para la palabra "INICIO", que se compone de su patrón (que es la misma palabra) y el código C que se va a ejecutar cuando se identifique en la entrada. El código muestra en la salida estándar el aviso "reconocido INICIO".
    - Reglas similares a la anterior para las palabras "FIN", "VECTOR", "ENTERO" y "LOGICO" .

ejemplo1.l

```
%{
#include <stdio.h>          /* para utilizar printf en la
                               sección de reglas */
%}
%option noyywrap            /* para procesar sólo 1
                               fichero */


%%
INICIO   { printf("reconocido INICIO\n"); }
FIN      { printf("reconocido FIN\n"); }
VECTOR   { printf("reconocido VECTOR\n"); }
ENTERO   { printf("reconocido ENTERO\n"); }
LOGICO   { printf("reconocido LOGICO\n"); }
%%
```

La sección de funciones de usuario (I)

- La sección de funciones de usuario se copia literalmente en el fichero de salida.
- Esta sección habitualmente contiene las funciones escritas por el usuario para ser utilizadas en la sección de reglas, es decir, funciones de soporte.
- En esta sección también se incluyen las funciones de Flex que el usuario puede redefinir, por ejemplo, la función **yywrap()** se situaría en esta sección.
- La función **yylex()** generada por Flex, tiene que ser invocada desde algún punto, habitualmente desde otro módulo C. Pero para realizar pruebas de la función **yylex()** se puede incorporar una función **main()** en la sección de funciones de usuario dentro del fichero de especificación Flex. Una versión muy simple sería:

```
int main()
{
        return yylex();
}
```

Esta llamada única a **yylex()** permite realizar el análisis hasta encontrar el fin de la entrada siempre que en ningún fragmento de código C asociado a los patrones aparezca una sentencia **return** que haga terminar a **yylex()**. En ese caso el **main()** sería distinto como se mostrará más adelante.

La sección de funciones de usuario (II)

- Se completa el fichero **ejemplo1.l** incluyendo una función **main()** en la sección de funciones de usuario.

ejemplo1.l

```
%{
#include <stdio.h>          /* para utilizar printf en la
                               sección de reglas */
%}
%option noyywrap            /* para procesar sólo 1
                               fichero */


%%
INICIO   { printf("reconocido INICIO\n"); }
FIN      { printf("reconocido FIN\n"); }
VECTOR   { printf("reconocido VECTOR\n"); }
ENTERO   { printf("reconocido ENTERO\n"); }
LOGICO   { printf("reconocido LOGICO\n"); }
%%

int main()
{
        return yylex();
}
```

Solución del primer ejemplo (I)

Creación del ejecutable

- A partir del fichero de especificación Flex **ejemplo1.l** se genera el ejecutable de la siguiente manera:
  - Compilar la especificación Flex:

    flex ejemplo1.l ⟶ se crea el fichero lex.yy.c

  - Generar el ejecutable:

    gcc -Wall -o ejemplo1 lex.yy.c ⟶ se crea el fichero ejemplo1

  - Ejecutar:

    ./ejemplo1

Solución del primer ejemplo (II)

Realización de pruebas

- Para probar el funcionamiento del analizador léxico implementado en el primer ejemplo, se arranca el ejecutable y se realizan las siguientes pruebas:

ENTRADA Y SALIDA	PROCESO
cadena cualquiera cadena cualquiera	No se identifica ninguna cadena. Se copia la entrada en la salida.
INICIO reconocido INICIO	Se identifica la cadena "INICIO". Se muestra un mensaje de aviso.
FIN reconocido FIN	Se identifica la cadena "FIN". Se muestra un mensaje de aviso.
VECTORENTERO reconocido VECTOR reconocido ENTERO	Se identifican los cadenas "VECTOR" y "ENTERO". **Observar que ambas cadenas se reconocen aunque no están separadas.** Se muestran los mensajes de aviso correspondientes.
^D	Fin de la entrada (en linux). El analizador termina su ejecución.

Texto azul: entrada del usuario
Texto rojo: respuesta del analizador

Los patrones de Flex (I)

Descripción

- Los patrones de Flex son:
  - el mecanismo para representar el lenguaje regular.
  - una extensión de las expresiones regulares.
- Los patrones están formados por:
  - caracteres "normales" que se representan a sí mismos.
  - metacaracteres que tienen un significado especial.
- Para utilizar un metacarácter como carácter "normal" hay que ponerlo entre comillas. Por ejemplo, como el asterisco es un metacarácter, si se quiere reconocer el token asterisco, hay que definirlo como "*".
- En los siguientes apartados se describen algunos de los metacaracteres de Flex.
- Un mismo lenguaje se puede expresar con distintos patrones.

Metacaracteres (I)

.	Representa cualquier carácter exceptuando el salto de línea.

[ ]	Representa cualquiera de los caracteres que aparecen dentro de los corchetes. Para indicar un rango de caracteres se utiliza el símbolo menos "-".

- `[xyz]` representa una "x", una "y" o una "z"
- `[abj-oZ]` representa una "a", una "b" cualquier letra de la "j" a la "o", o una "Z"
- `[ \t]` representa el espacio y el tabulador
- `[0-9]` representa los dígitos del 0 al 9
- `[a-z]` representa las letras minúsculas
- `[A-Z]` representa las letras mayúsculas
- `[a-zA-Z]` representa las letras minúsculas y las mayúsculas
- `[0-9a-zA-Z]` representa los dígitos del 0 al 9, las letras minúsculas y las mayúsculas

^	Niega los caracteres que le siguen.

- `[^A-Z]` cualquier carácter excepto una letra mayúscula.
- `[^ \t\n]` cualquier carácter excepto blanco, tabulador o salto de línea.

Metacaracteres (II)

*	Indica 0 ó más ocurrencias de la expresión que le precede.

- `ab*` representa todas las palabras que empiezan por una "a" seguida de 0 o más "b", por ejemplo "a", "ab", "abb", "abbb".
- `[a-zA-Z][a-zA-Z0-9]*` representa todas las palabras que empiezan por una letra minúscula o mayúscula seguida de 0 o más letras o dígitos, como por ejemplo "v1", "indice", "maximo", "D".

+	Indica 1 ó más ocurrencias de la expresión que le precede.

- `x+` representa todas las palabras formadas por "x", por ejemplo "x", "xx", "xxx".
- `[0-9]+` representa números de uno o más dígitos, por ejemplo "12", "465", "0".

?	Indica 0 ó 1 ocurrencia de la expresión que le precede.

- `"-"?[0-9]+` representa números de uno o más dígitos con o sin signo menos.

Metacaracteres (III)

{m} Indica m ocurrencias de la expresión que le precede.

- `a{4}` representa la palabra que tienen 4 letras "a".

{m,} Indica m ó más ocurrencias de la expresión que le precede.

- `x{3,}` representa todas las palabras formadas por 3 o más "x"

{m,n} Indica entre m y n ocurrencias de la expresión que le precede.

- `("-"?[0-9]+){1,2}` representa todas las secuencias de 1 ó 2 números de uno o más dígitos con o sin signo menos.

Metacaracteres (IV)

| | Identifica la expresión que le precede o la que le sigue.

- `A|B` representa la letra "A" o la letra "B". Este patrón se comporta de la misma manera que el patrón `[AB]`.

"..." Representa lo que esté entre las comillas literalmente. Los metacaracteres pierden su significado excepto "\". Para expresar el carácter \ se utiliza "\\". Para representar el carácter " se utiliza "\"".
El metacarácter \ si va seguido de una letra minúscula se asume que es una secuencia de escape de `C`, como por ejemplo el tabulador \t.

- "`/*`" representa a la agrupación de caracteres "/*".

( ) Agrupan expresiones.

- (ab|cd)+r representa "abr", "ababr", "cdr", "cdcdr", “abcdr”, etc.

Metacaracteres (V)

{ nombre } Un nombre encerrado entre llaves significa la expansión de la definición de "nombre".
En la sección de definiciones se pueden definir patrones y asignarles un nombre. Por ejemplo:

```
DIGITO   [0-9]
LETRA    [a-zA-Z]
```

Posteriormente, cualquier aparición de `{DIGITO}` se sustituye por la expresión regular `[0-9]`, y `{LETRA}` por `[a-zA-Z]`.

La expresión regular de todas las palabras que empiezan por una letra minúscula o mayúscula seguida de 0 o más letras o dígitos:

```
[a-zA-Z]([0-9]|[a-zA-Z])*
```

es idéntica a:

```
{LETRA} ({DIGITO}|{LETRA})*
```

Cómo se identifican los patrones en la entrada (I)

- El analizador busca en la entrada cadenas de caracteres que concuerden con alguno de los patrones definidos en la sección de reglas.

- La función generada por Flex siempre intenta buscar concordancia de manera que se consuma el máximo número de caracteres de la entrada.
  Por ejemplo, si se definen las reglas:

```
INICIO  { return TOK_INICIO; }
FIN     { return TOK_FIN; }
[A-Z]+  { return TOK_ID;}
```

  La entrada INICIOFIN se identificará como TOK_ID porque el análisis de la entrada se realiza de la siguiente manera:

  - Desde la primera "I" de la cadena de entrada hasta la "O", concuerdan dos patrones, el de TOK_INICIO y el de TOK_ID.
  - Cando se lee la letra "F", en ese momento se descarta el patrón correspondiente a TOK_INICIO, y se selecciona el patrón de TOK_ID que es más largo.

Cómo se identifican los patrones en la entrada (II)

- Si hay concordancia con varios patrones, siempre se elige el patrón que esté situado primero en la sección de reglas dentro del fichero de especificación Flex. Por lo tanto, **es determinante el orden en que se colocan las reglas.**
  Por ejemplo, si se definen las reglas:

```
[A-Z]+        { return TOK_ID;}
INICIO        { return TOK_INICIO; }
FIN           { return TOK_FIN; }
```

La entrada INICIO se identificará como TOK_ID porque hay concordancia con dos patrones, el de TOK_INICIO y el de TOK_ID, pero el patrón de TOK_ID aparece antes en el fichero de especificación.

Al procesar con Flex estas reglas, se muestra un mensaje en el que se indica que las reglas segunda y tercera nunca se van a identificar (Se recomienda construir un analizador de prueba).

Cómo se identifican los patrones en la entrada (III)

- En el ejemplo anterior, para que se identifiquen las palabras reservadas INICIO y FIN correctamente, y no como identificadores, se deben colocar sus correspondientes reglas antes de la regla de los identificadores, de la siguiente manera:

```
INICIO          { return TOK_INICIO; }
FIN             { return TOK_FIN; }
[A-Z]+          { return TOK_ID;}
```

- Cuando se ha determinado el patrón que concuerda con la entrada, además de ejecutarse el código C asociado al patrón, se producen las siguientes acciones:
  - el texto de la cadena se almacena en la variable de Flex **yytext** que es un **char***
  - la longitud de la cadena se almacena en la variable de Flex entera **yyleng**

Ficheros de entrada/salida de Flex

- **FILE* yyin**
  - Es el fichero de entrada, del que lee la función **yylex()**
  - Por defecto es la entrada estándar.
  - El usuario debe realizar la apertura de yyin antes de que se inicie el análisis, es decir, antes de invocar a la función yylex().

- **FILE* yyout**
  - Es el fichero de salida, en el que se escribe la regla por defecto (cuando no concuerda ningún patrón se copia la entrada en la salida)
  - Por defecto es la salida estándar.
  - El usuario debe realizar la apertura de yyout antes de que se inicie el análisis, es decir, antes de invocar a la función yylex().

Modificación del primer ejemplo

- Modificar el primer ejemplo para que la entrada al analizador se realice a través de un fichero. Para ello se deberá modificar el main() de la especificación Flex de manera que el primer parámetro de la llamada al ejecutable sea el nombre del fichero de entrada.

- El nombre de la nueva especificación Flex será “ejemplo1_b.l”