Práctica 2
Manejo Avanzado de *Flex*

Jorge Bernad, Elvira Mayordomo, Mónica Hernández, José Manuel Colom

Tareas

1. Estudia la sección sobre las *condiciones de arranque* en el libro *flex & bison* (páginas 28 a 31) y el capítulo 9 del manual de *Flex* disponible en moodle.

2. Lee la introducción de esta práctica y realiza los ejercicios propuestos.

3. Elabora la memoria de la práctica y entrégala junto con los ficheros fuente según el Procedimiento de Entrega de Prácticas explicado en la Introducción a las Prácticas de la Asignatura. La fecha tope de entrega será hasta el día anterior al comienzo de la Práctica 3.

Nota: El incumplimiento de las normas de entrega se reflejará en la calificación de la práctica.

Se recuerda especialmente lo siguiente:

- Crea un directorio que contenga exclusivamente el fichero con la memoria en formato *PDF*, los ficheros fuente con tu código *.l* de *Flex*, y los de prueba (*.txt* de texto). No usar subdirectorios.

- Accede al directorio con tus ficheros y ejecuta el comando

```
zip nipPr2.zip *.*
```

  donde *nip* es el identificador personal.

- En caso de que el fichero resultante tenga un tamaño mayor de 1024 KB deberás dividir el fichero `nipPr2.zip` en varios ficheros de tamaño 1024KB con el comando de linux

```
split -b 1m nipPr2.zip nipPr2.zip.
```

Introducción

El objetivo principal de esta práctica es aprender a desarrollar analizadores léxicos en *Flex* más sofisticados, profundizando en el manejo de lo que se conoce como *condiciones de arranque*. Las *condiciones de arranque* no son imprescindibles, ya que siempre se pueden emular utilizando código *C* en las acciones de los patrones. No obstante, su uso facilita mucho el desarrollo de los programas en *Flex,* ya que ayudan a estructurar conjuntos de patrones/acciones en función de un contexto determinado o condiciones previas.

Ejercicio 1

El objetivo de este ejercicio es generar un programa usando *Flex* para escanear ficheros y detectar algunos tipos de virus informáticos. Una forma de detectar que un fichero está infectado por un virus determinado es buscar un patrón de bytes que identifican al virus. Ese patrón de bytes es la llamada *firma* del virus[1]. En el fichero adjunto a la práctica *virussignatures.txt*[2] aparecen unas 1.300 firmas de distintos virus conocidos. Este fichero se compone de líneas de texto con el siguientes formato:

1. nombre del virus: compuesto por cualquier carácter, excepto el carácter '=';
2. el carácter '=';
3. la firma del virus expresada con dígitos en hexadecimal: la firma está compuesta por los caracteres del 0 al 9 y las letras de la A a la F (tanto en mayúsculas como en minúsculas). Cada dos dígitos en hexadecimal consecutivos representan un byte de la firma del virus. Por ejemplo, si en el fichero apareciese $32A426$, significa que la firma del virus está compuesta por tres bytes cuyos códigos ascii en hexadecimal son 32, $A4$ y 26, respectivamente.

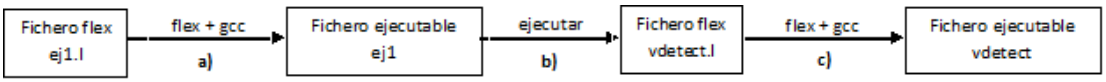

Figura 2.1: Proceso para generar el programa que detecta virus.

El proceso para generar el programa que detecte las firmas contenidas en el fichero *virussignatures.txt* será el siguiente: primero (Figura 2.1 paso a), deberéis generar un fichero de *Flex* de nombre *ej1.l* para generar un programa (*ej1*) que use la información de un fichero con firmas de virus con el formato explicado en 1-3; al ejecutarse *ej1* (Figura 2.1 paso b) se generará un fichero de *Flex* (*vdetect.l*) con tantas reglas como firmas de virus aparecen en el fichero. El patrón de cada una estas reglas será la secuencia de bytes de la

[1]Más información en `https://www.kaspersky.com/blog/signature-virus-disinfection/13233/`
[2]Fichero obtenido de `http://codes-sources.commentcamarche.net/source/download/21418/892294`

firma de un virus, y la acción asociada será simplemente mostrar por pantalla el nombre del virus; y tercero (Figura 2.1 paso c), una vez creado automáticamente este fichero de *Flex*, se puede compilar[3] para generar un programa que sea capaz de escanear cualquier fichero detectando los virus. Si el fichero está infectado, el programa nos mostrará el nombre del virus; y si no está infectado, no mostrará ningún mensaje.

Observación: como solución a este ejercicio, se presentará únicamente el fichero *ej1.l*. No es necesario que presentéis el fichero de *Flex* que contiene todas las reglas identificando las firmas de los virus.

Ejercicio 2

Muchos editores de código mutlilenguaje (como Notepad++, Vim) utilizan patrones para identificar automáticamente el lenguaje de programación en el que está escrito un fichero, y de esta forma, proporcionar al usuario una mejor experiencia de uso. El objetivo de este ejercicio es generar un programa que identifique ficheros escritos en Java o en C++, o en otras palabras, un programa que clasifique un código en Java ó C++. Con este objeto, tendremos definidos dos contadores que representarán la puntuación obtenida por el código para Java y para C++. Aquel lenguaje que sume más puntos será el elegido. El total de la puntuación se halla mediante las siguientes reglas:

- Si en el fichero aparece la cadena `#include`, le sumaremos 10 puntos al contador de C++ (sólo para el primer include).
- Si en el fichero aparece al inicio de una línea (con posibles espacios en blanco previos) la cadena `import`, le otorgaremos 10 puntos más al contador de Java (sólo para el primer import).
- Por cada aparición de `String` (la primera S en mayúscula, el resto en minúsculas) seguido de posibles espacios en blanco y de una cadena de letras/dígitos y el carácter '_' que no empieza por dígito, se sumará un punto al contador de Java, hasta un máximo de 5 puntos adicionales.
- Si aparece `int`, `float`, `char` seguido de posibles espacios en blanco, del carácter '&', más posibles blancos y de una cadena de letras/dígitos y el carácter '_' que no empieza por dígito, se sumará un punto al contador de C++, hasta un máximo de 5 puntos adicionales.

Hay que tener en cuenta que las reglas anteriores sumarán puntos a los contadores siempre y cuando los patrones ocurran fuera de los dos estilos de comentarios de Java y C++

[3] Al compilar con flex un fichero con muchas reglas puede aparecer un mensaje de error "`input rules are too complicated`". Este error se puede evitar compilando flex con la opción `-Ca`. Más información sobre la opción de compilación en la ayuda de flex (`man flex`).

`/*...*/` `//... <FinLinea>`

o fuera de la declaración de una constante de cadena,

`"..."`

Para simplificar el problema, supondremos que dentro de las constantes de cadena no puede aparecer el caracter ".

Implementad un escáner con *Flex* (el código flex se guardará en un fichero llamado *ej2.l*) que analice un código y nos informe de la puntuación obtenida y la clasificación final, `JAVA` ó `C++`. La salida del programa constará de tres líneas con el formato,

```
PJ:<puntuación Java>
PC:<puntuación C++>
CLASIFICADO:<uno de JAVA ó C++>
```