Práctica 2
Manejo Avanzado de *Flex*

Jorge Bernad, Elvira Mayordomo, Mónica Hernández, José Manuel Colom, Carlos Bobed, Gregorio de Miguel

Tareas

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

- Verifica que todos tus ficheros fuente (*.l*) contienen como comentario en sus primeras líneas los NIAs y nombres de los autores. Todos los programas deberán estar debidamente documentados.
- Verifica que los ficheros que vas a presentar compilan y ejecutan correctamente en lab000.
- Crea un fichero comprimido .zip llamado

  `niaPr2.zip`

  donde *nia* es el identificador personal. Este fichero comprimido **debe contener 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.
- Utiliza el enlace a una tarea de moodle disponible en la sección "Prácticas de Laboratorio" para entregar el fichero `niaPr2.zip`

Introducción

Los objetivos de esta práctica son:

- 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.
- Aplicar los conocimientos teóricos de Automátas Finitos para desarrollar programas.

Ejercicio 1

La falta de comunicación entre distintas partes de un equipo produce grandes pérdidas de tiempo y dinero. Aunque el caso que presentamos en este ejercicio no es real, sí que está basado en hechos reales. Como las mejores películas de sobremesa.

Dos equipos de desarrolladores, pongamos el Equipo A y el Equipo B, están implementando un importante software en C. En la implementación se llama miles de veces a una pequeña función auxiliar llamada `ups` con dos parámetros de distinto tipo, digamos que uno es un array de enteros y el otro es un entero. Como es una pequeña función, cada equipo la ha implementado por su cuenta. También hay versiones sobrecargadas de esta función con cero, uno, tres o más parámetros. Al juntar el código de los dos equipos se dieron cuenta que el proyecto no compilaba por un error muy tonto: mientras que el Equipo A había considerado que el primer parámetro de `ups` era un array de enteros y el segundo de tipo entero, el Equipo B lo había considerado al revés. La solución era sencilla. Uno de los equipos tenía que cambiar el orden de los parámetros en cada llamada a la función `ups`. Ese equipo también marcaría los comentarios dentro del código donde se nombraba a cualquier versión de la función `ups` como "NO ACTUALIZADO" para evitar documentación incorrecta.

Finalmente se decide cambiar el código del Equipo A porque, además de ser sintácticamente correcto, cumplía las siguientes condiciones que facilitaban las modificaciones utilizando Flex:

- En las llamadas a las funciones, entre el nombre de la función y el paréntesis que marca el inicio de la lista de parámetros solo hay cero o más espacios en blanco.
- No hay comentarios dentro de las llamadas de las funciones. Por ejemplo, no aparecen llamadas como:

  ```
  ups(/*comemt par 1*/a1, /*coment par 2*/a2)
  ```

- En las llamadas a las funciones, los únicos paréntesis y comas que aparecen son los necesarios para hacer la llamada y separar los parámetros. No hay llamadas como:

  ```
  ups(f(a1,a2), b2)
  ```

  Por tanto, cada llamada a una función la podemos ver como una lista de bloques de parámetros separados por comas:

f(p_1 , p_2 ,...)

donde cada bloque p_i no contiene ni comas ni paréntesis. **Nótese que el objetivo es intercambiar el bloque p_1 y p_2 cuando la función se llama `ups` y tenga exactamente dos parámetros**. El intercambio incluye todos los caracteres que aparezcan en cada bloque (posibles saltos de línea, espacios en blanco, etc).

- Los nombres de variables y funciones están compuestos por letras minúsculas, mayúsculas y/o dígitos. Ninguna variable se llama `ups`, aunque sí hay variables, y también funciones, cuyos nombres contienen `ups` (como, por ejemplo, `cups`, `ups1`, ...)

Implemente con Flex (fichero *ej1.l*) un analizador léxico que dado un texto modifique las llamadas a la función `ups` como se ha explicado. Supondremos que en el texto no está la declaración de la función, solo hay llamadas. En los comentarios en los que aparezca la palabra `ups` se incluirá el texto `NO ACTUALIZADO` al final de un comentario de tipo `//` (antes del carácter fin de línea `\n` con el que termina el comentario), o antes del último asterisco que cierra un comentario de tipo `/*...*/`. Dentro de un comentario no se intercambiarán los parámetros. Por ejemplo, el comentario

```
// si x>0, ups(v,x) no está definido
```

se deberá modificar por

```
// si x>0, ups(v,x) no está definidoNO ACTUALIZADO
```

Ejemplo de entrada y la salida esperada:

Entrada:

```
/* un comentario */

//comentario con ups
if ( ups(p1,p2)&&ups(t1,t2+t3)) {
    a = b + c;
}

/*5>ups(a1,a2)>0*/
ups    (a1   ,
    a2 );

ups(r1);

ups(a1,a2,ups1,ups2);

//otro comentario con cups
cups(a1,a2);

/*
 * Comentario con ups
 * // ups
 ***/
ups()
```

Salida:

```
/* un comentario */

//comentario con upsNO ACTUALIZADO
if ( ups(p2,p1)&&ups(t2+t3,t1)) {
        a = b + c;
}

/*5>ups(a1,a2)>0NO ACTUALIZADO*/
ups    (
    a2  ,a1   );

ups(r1);

ups(a1,a2,ups1,ups2);

//otro comentario con cups
cups(a1,a2);

/*
 * Comentario con ups
 * // ups
 **NO ACTUALIZADO*/
ups()
```

Observaciones: será necesario guardar los caracteres de un bloque de parámetros en una variable de tipo `char*` para su posterior intercambio. La librería `string.h` contiene la función `strdup` para hacer una copia de una cadena de caracteres. Por ejemplo, si `p1` es una variable de tipo `char*`, la siguiente instrucción

```
p1 = strdup(yytext);
```

crea una copia de los caracteres de la variable `yytext` en `p1`.

Ejercicio 2

En la película *La jungla de cristal 3* un misterioso malvado que se hace llamar Simon propone resolver una serie de acertijos al detective John McClean y a Zeus Carver, un electricista de Harlem, para evitar la explosión de unas bombas diseminadas por todo Nueva York. Uno de los acertijos consiste en desactivar una de las bombas poniendo sobre una plataforma un bidón con exactamente 4 galones de agua. El problema consiste en que para conseguir los 4 galones de agua solo disponen de un bidón de 5 galones y otro de 3 que pueden rellenar y vaciar en una fuente. Los protagonistas solo pueden realizar la siguientes acciones, denotadas por $\{V, v, R, r, T, t\}$:

- Acciones V/v: se puede vaciar el contenido de un bidón en el suelo. Con V y v denotaremos el vaciado del bidón grande y pequeño, respectivamente. Supondremos que no se permiten secuencias de acciones en las que se vacíe un bidón que ya está vacío. Tampoco estará permitida una secuencia de acciones que en algún momento deje vacíos los dos bidones[1].
- Acciones R/r: rellenar de agua un bidón hasta llenarlo completamente (R bidón grande, r bidón pequeño). No se permite cualquier secuencia de acciones en la que se rellene un bidón que ya está lleno. Igualmente, no estará permitida cualquier secuencia de acciones que deje en algún momento los dos bidones llenos.
- Acciones T/t: trasvasar el agua de un bidón a el otro (T del bidón grande al pequeño, y t viceversa). El trasvase finaliza cuando el bidón de destino esté lleno o el bidón fuente quede vacío, lo que ocurra antes. No se puede trasvasar agua ni desde un bidón vacío, ni a un bidón lleno. Además, a nuestros héroes solo les estará permitido o realizar trasvases del bidón grande al pequeño o trasvases del pequeño al grande. Esto es, la secuencia de acciones, aparte de vaciados y rellenados, solo puede contener o Tes o $t$es.

Supondremos que siempre empezamos con los dos bidones vacíos. Un ejemplo de secuencia de acciones que permiten obtener un bidón con 4 galones es $RTvTRT$. Nótese que en esta secuencia de acciones solo hay trasvases del bidón grande al pequeño, además de respetar el resto de restricciones, y por tanto, está permitida.

Ejemplos de secuencias no permitidas son (se marca en negrita y subrayada la primera acción que invalida la secuencia):

- Vaciar un bidón ya vacío: $RTv\underline{\mathbf{v}}T$, $rt\underline{\mathbf{v}}$, $\underline{\mathbf{v}}RT$.
- Dejar los dos bidones vacíos: $RTV\underline{\mathbf{v}}$, $rt\underline{\mathbf{V}}R$.
- Rellenar un bidón lleno: $R\underline{\mathbf{R}}T$, $RT\underline{\mathbf{r}}$.
- Dejar los dos bidones llenos: $r\underline{\mathbf{R}}$, $RT\underline{\mathbf{R}}v$.
- Trasvasar desde un bidón vacío: $\underline{\mathbf{T}}RV$, $rt\underline{\mathbf{t}}$, $rtR\underline{\mathbf{t}}$.
- Trasvasar a un bidón lleno: $RT\underline{\mathbf{T}}V$, $rtrt\underline{\mathbf{t}}$.
- Mezcla de T y t: $RTvr\underline{\mathbf{t}}$.

[1] Estas restricciones, junto con las descritas en los siguientes puntos, no suceden en la película, pero no desvirtúan el problema.

Crear un programa con Flex (fichero *ej2.l*) que dado un texto, con una secuencia de acciones por línea, marque cada línea de la siguiente forma:

- si la secuencia de acciones de la línea está permitida y se obtiene un bidón con 4 galones utilizando T se escribirá al inicio de una línea dos signos de sumar, `++`, seguidos de la secuencia;
- si la secuencia de acciones está permitida y se obtiene un bidón con 4 galones utilizando t se pondrán al inicio de la línea dos signos menos, `--`, seguidos de la secuencia;
- en cualquier otro caso, si la secuencia no está permitida o no se consiguen 4 galones, se pondrán al inicio dos asteriscos, `**`, seguidos de la secuencia.

Se puede suponer que en el texto de entrada solo aparecen los seis caracteres que denotan las acciones y los saltos de línea.
Ejemplo:

Entrada	**Salida**
`RTvTRT`	`++RTvTRT`
`RTvTRTr`	`**RTvTRTr`
`RRTvTRT`	`**RRTvTRT`
`RTtRT`	`**RTtRT`
`RTvTRTv`	`++RTvTRTv`
`RTvTRTvR`	`**RTvTRTvR`
`vRTvTRT`	`**vRTvTRT`

Responda razonadamente en la memoria, basándose en la teoría de autómatas finitos, a las siguientes preguntas:

1. Si el objetivo hubiese sido tener en un bidón 4 galones y en el otro 1 galón, ¿habría sido posible desactivar la bomba?
2. Incluya en la memoria una solución utilizando únicamente t.

Observaciones: Plantee dos AFDs para solucionar el problema: uno para detectar las secuencias correctas conteniendo solo T, y otro para las secuencias con solo t. La funcionalidad de JFlap para transformar un AFD en una expresión regular facilita la resolución de este ejercicio. Recuerde que las expresiones regulares que proporciona JFlap se adaptan a la sintaxis explicada en las clases de teoría, pero no a la sintaxis de Flex.