Teoría de la Computación

Grado en Ingeniería Informática

- Prácticas de Laboratorio - *

Jorge Bernad
email: jbernad@unizar.es
José Manuel Colom Piazuelo
email: jm@unizar.es
Mónica Hernández
email: mhg@unizar.es
Elvira Mayordomo Cámara
email: elvira@unizar.es
Dpto. de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2018-2019

*Material elaborado parcialmente a partir de los guiones de prácticas mantenidos por los profesores Gregorio de Miguel Casado (Teoría de la Computación curso 2011-2012), Pedro Álvarez, Rubén Béjar y Jorge Júlvez para la asignatura *Lenguajes Gramáticas y Autómatas* de la titulación *Ingeniería de Informática* (122) del plan de estudios BOE 1-2-1995.

Introducción a las Prácticas de la Asignatura
Entorno de Trabajo y Entrega de Prácticas

Las prácticas de la asignatura Teoría de la Computación abordan aspectos de implementación de reconocedores para expresiones regulares (análisis léxico) y reconocedores de lenguajes caracterizados con gramáticas (análisis sintáctico) mediante las herramientas Unix *Flex* y *Bison*, respectivamente. El aprendizaje de estas herramientas es de interés general, ya que permite abordar, con posterioridad, la construcción de compiladores, programas para la traducción/migración entre formatos de ficheros y similares.

Entorno de Trabajo

Las prácticas de Teoría de la Computación se realizan en Hendrix, cuya dirección es **hendrix-ssh.cps.unizar.es**. Podeis abrir sesión en cualquier terminal de Linux y utilizar el editor *gedit* para editar vuestros ficheros fuente. Para compilar los programas, hay que compilar los programas usando los ejecutables correspondientes a *Flex* y *Bison.*
En las siguientes URLs podeis encontrar todo lo relacionado a estas herramientas.

- http://flex.sourceforge.net/
- http://www.gnu.org/software/bison/

También se utilizarán como material de referencia el siguiente libro y manuales:

- *flex & bison*, John Levine, Ed. O'Reilly.
- Manual de flex versión 2.5 (disponible en moodle)
- Manual de bison versión 1.27 (disponible en moodle)

Entrega de Prácticas

Las prácticas se realizarán de forma **individual**.

Para cada una de las cuatro prácticas se deberá entregar un paquete *.zip* que contenga una memoria en formato *PDF* y los ficheros fuente y de prueba para cada ejercicio planteado. Los siguientes apartados detallan los contenidos de la memoria, el procedimiento para empaquetar en un fichero *.zip* los ficheros para la entrega y, finalmente el mecanismo de entrega mediante la orden *someter* en *Hendrix*.

El incumplimiento de las normas establecidas en este apartado para el formato de la memoria y/o ficheros se reflejará en la calificación de la práctica.

Las copias o plagios que se detecten en las memorias y/o programas supondrán un suspenso directo de la parte práctica de la asignatura.

Formato de la Memoria

- **Portada:** Número de Práctica, Grupo y Autor. Ejemplo:

> **Grupo Miércoles 12:00-14:00 semanas B**
> **– Práctica 1 –**
> **Autor:** Al Anturing

- **Una sección para cada ejercicio resuelto.** Razona todas las decisiones de implementación que has tomado en la elaboración de tu código e incluye las pruebas de ejecución realizadas. Ejemplo:

> **Ejercicio 1:**
>
> **1. Resumen**
>
> He creado el patrón X para poder reconocer Y
>
> ...
>
> **2. Pruebas**
>
> Para la entrada Z he obtenido la salida W
>
> ...

Nota: el formato del fichero de la memoria deberá ser *PDF*.

Empaquetado de los Ficheros

- Accede a tu cuenta en *Hendrix* con un terminal de *Linux*:

```
ssh -X usuario@hendrix-ssh.cps.unizar.es
```

- Verifica que todos tus ficheros fuente (*.l* de *Flex* y *.y* de *Bison*) contienen en sus primeras líneas número de práctica y ejercicio así como el NIP y nombre del autor. Todos los programas deberán estar debidamente documentados.
- Crea un directorio que contenga exclusivamente el fichero con la memoria en formato *PDF*, los ficheros fuente con tu código (*.l* de *Flex* y *.y* de *Bison*) y los de prueba (*.txt* de texto). No usar subdirectorios.
- Accede al directorio con tus ficheros y ejecuta la orden

```
zip nipPrX.zip *.*
```

  donde *nip* es el identificador personal y *X* es el número de práctica (1,2,3 ó 4).
- En caso de que el fichero resultante tenga un tamaño mayor de 1024 KB deberás dividir el fichero `nipPrX.zip` en varios ficheros de tamaño 1024KB con la orden de linux

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

  Se crearán los ficheros `nipPrX.zip.aa`, `nipPrX.zip.ab`, ..., cada uno de tamaño máximo 1024KB.

Entrega con *someter* en *Hendrix*

- Una vez que ya tengas el fichero .zip, en tu cuenta de *Hendrix* ejecuta la orden:

```
someter -v tc_18 nipPrX.zip
```

  o si el fichero `nipPrX.zip` tiene más de 1024 KB, somete cada uno de los ficheros

```
someter -v tc_18 nipPrX.zip.aa
someter -v tc_18 nipPrX.zip.ab
...
```

- La fecha tope de entrega para cada una de las prácticas será hasta el día anterior a la sesión en la que comience la siguiente práctica. Para la Práctica 4 se concretará una fecha específica.

Práctica 1: Introducción a Flex

Tareas

1. Aprende a acceder y trabajar con tu cuenta en Hendrix.
2. Lee las págs 1-4 del libro *flex & bison*, John Levine, Ed. O'Reilly.
3. Lee la introducción de esta práctica y realiza los ejercicios 1 a 4 propuestos en la parte A (primera sesión).
4. Lee la introducción de la parte B y realiza los ejercicios 5 a 8 propuestos en la parte B (segunda sesión).
5. 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 (página 3 de este documento). La práctica 1 tiene 2 sesiones. La fecha tope de entrega será hasta el día anterior al comienzo de la Práctica 2 (tercera sesión).

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

Introducción

El objetivo principal de esta práctica de la asignatura es familiarizarse con la herramienta de creación de analizadores léxicos *Flex*. Para ello, se propone la creación con dicha herramienta de una serie de pequeños procesadores de texto.

Las prácticas de Teoría de la Computación se realizan en **hendrix-ssh.cps.unizar.es**. Para compilar los programas, hay que compilar los programas usando los ejecutables correspondientes a *Flex* y *Bison*. Ejemplo (*Flex*)[1]:

```
flex nombre_fichero_fuente.l
gcc lex.yy.c -lfl -o nombre_ejecutable
./nombre_ejecutable <fichentrada >fichsalida
```

[1] En sistemas Mac OS X, si da problemas al enlazar, sustituir la opción -lfl por -ll, esto es, compilar con *gcc lex.yy.c -ll -o nombre_ejecutable*

El alfabeto Σ que maneja *Flex* está formado por los caracteres manejables por el sistema (p. ej. todos los símbolos del código ASCII). En las prácticas se trabaja con letras (distinguiendo entre mayúsculas y minúsculas, no se usa la ñ), números, 8 signos de puntuación (! , . : ; ( ) ?), 3 separadores (espacio, tabulador y salto de línea) y 23 símbolos visibles ($''$ # \$ % & $'$ $*$ $+$ $-$ / $<$ $=$ $>$ @ [\ ] $\hat{}$ _ { | } $\tilde{}$). No es necesario preocuparse de símbolos no visibles, letras acentuadas o cualquier otro carácter que no hayamos mencionado aquí.

Parte A: Introducción al manejo de *Flex*

Ejercicio 1

Disponemos de un fichero de texto con las cuentas de mail de una serie de usuarios. Escribe un programa con *Flex* de nombre *ej1.l* que sustituya cualquier correo de *hotmail* por *gmail*. Esto es, cada vez que aparezca la cadena *@hotmail* la cambie a *@gmail*

Ejemplo:

Entrada:

```
perico@hotmail.com  ana@unizar.es
sab@hotmail.com  javier@garcia.com
maria@hotmail.com  modrego@rm.edu
jose@unizar.es
```

Salida:

```
perico@gmail.com  ana@unizar.es
sab@gmail.com  javier@garcia.com
maria@gmail.com  modrego@rm.edu
jose@unizar.es
```

Ejercicio 2

Elabora un programa en *Flex* de nombre *ej2.l* que permita contar el número de usuarios de correo de hotmail, esto es, el número de apariciones de la cadena *@hotmail*

Ejemplo:

Entrada:

```
perico@hotmail.com  ana@unizar.es
sab@hotmail.com  javier@garcia.com
maria@hotmail.com  modrego@rm.edu
jose@unizar.es
```

Salida:

```
perico@hotmail.com  ana@unizar.es
sab@hotmail.com  javier@garcia.com
maria@hotmail.com  modrego@rm.edu
jose@unizar.es
Total de usuarios: 3
```

Ejercicio 3

Escribe un programa con *Flex* de nombre *ej3.l* que sustituya cualquier email de la Universidad de Zaragoza por un correo del mismo usuario de gmail. Es decir, se debe sustituir cualquier aparición de *@unizar.es* por *@gmail.com*

Responde razonadamente en la memoria a las siguientes preguntas:

- Con lo que te han explicado en la clase de prácticas, ¿qué ocurre si un usuario tiene como cuenta de correo *jrg@unizaroes.es*? ¿Se modifica adecuadamente el mail? ¿Por qué?
- Intenta entender las páginas 19 y 20 del libro *flex & bison* e indica dos formas distintas de solucionar el problema. Cambia el código del fichero *ej3.l* con una de las dos formas propuestas.

Ejercicio 4

Construye un programa en *Flex* de nombre *ej4.l* que modifique:

- todas las apariciones de un cifra (del 0 al 9) por el número siguiente al que representa la cifra;
- todas las apariciones de un salto de línea por dos saltos de línea.

Ejemplo:

Entrada:

```
Mi numero de telefono es 548271210
ponte en contacto con
el asistente 59 en 04 minutos.
```

Salida:

```
Mi numero de telefono es 659382321

ponte en contacto con

el asistente 610 en 15 minutos.
```

Observaciones:

- Recuerda que se puede usar la función *atoi* para transformar una cadena de caracteres en un número entero.

```
int n = atoi(s);
```

- Para escribir un número entero con *printf* se puede usar

```
printf("%d", n);
```

Parte B: Introducción al manejo de *Flex*

Introducción

El objetivo principal de esta parte es familiarizarse con la herramienta de creación de analizadores léxicos *Flex* y **aprender aspectos básicos sobre teoría de lenguajes y expresiones regulares**. Para ello, se propone la creación con dicha herramienta de una serie de pequeños procesadores de texto.

El cuadro 1.1 muestra algunas correspondencias entre la notación empleada en clase de teoría y la que se utiliza en *Flex* para trabajar con expresiones regulares (ERs).

Operación	**ERs Teoría**	**ERs Flex**
Concatena ERs	$\cdot$ ó {vacío}	{vacío} (yuxtaposición)
Unión de ERs	$+$	$\vert$
Cerradura de Kleene	*	$*$
Cerradura Positiva	$^+$	$+$
Paréntesis	( )	( )
Símbolos a,b,c,d,0,1,2	$\{a, b, c, d, 0, 1, 2\}$	$[abcd012]$ ó $[a - d0 - 2]$
Cadena vacía ó a	$\epsilon + a$	$a?$
Cadena vacía	ϵ	Sin equivalencia (ver $*$ ó ?)
Conjunto vacío	$\emptyset$	Sin equivalencia

Cuadro 1.1: Correspondencia entre notaciones

Ejercicio 5

Un fan de Juego de Tronos quiere llevar un recuento de los capítulos de la serie que ha visualizado y de los personajes que han fallecido en cada episodio. Para ello, procede a ir escribiendo en un fichero de texto llamado *got.txt* cuántos personajes de las casas Stark, Lannister y Baratheon fallecen a lo largo de la serie. Conforme va viendo episodios, el fan añade una única línea al fichero para cada episodio. El orden de visualización es un poco caótico, por lo que los episodios pueden aparecer desordenados. Cada línea del fichero contiene la temporada, el número de episodio de la temporada, y por cada personaje fallecido una letra indicando la casa a la que pertenecía (S para Stark, L para Lannister y B para Baratheon). Cada uno de estos campos se encuentran separados por un número indeterminado de espacios o tabuladores. Implemente un analizador léxico en *Flex* (fichero *ej5.l*) que analice el fichero de texto *got.txt* y genere otro con la siguiente información:

- cuántos episodios se han visto de cada temporada. La serie tiene 7 temporadas;
- total de personajes fallecidos en cada casa;
- casa con más muertes;

La salida estará compuesta por exactamente 11 líneas de texto con el formato:

```
T1:<episodios vistos temporada 1>
T2:<episodios vistos temporada 2>
T3:<episodios vistos temporada 3>
T4:<episodios vistos temporada 4>
T5:<episodios vistos temporada 5>
T6:<episodios vistos temporada 6>
T7:<episodios vistos temporada 7>
FS:<total de fallecidos Stark>
FL:<total de fallecidos Lannister>
FB:<total de fallecidos Baratheon>
MM:<casa con mas muertes>
```

Ejemplo:

Entrada:
```
1 5 B L S L
3 9 S S S S S
2 10 L B B
2 1
```

Salida:
```
T1:1
T2:2
T3:1
T4:0
T5:0
T6:0
T7:0
FS:6
FL:3
FB:3
MM: Stark
```

Ejercicio 6

Muchas órdenes de linux[2], como *egrep, grep, find, sed, etc*, tienen como parámetro de entrada una expresión regular. Por ejemplo, la orden *egrep*, en su versión más simple, tiene dos parámetros: el primero es una expresión regular entre comillas simples; el segundo es el nombre de un fichero de texto. Al ejecutar la orden, te muestra aquellas líneas del fichero que concuerdan con el patrón de la expresión regular. Por ejemplo, si tenemos el fichero *texto.txt*, al ejecutar la orden *egrep 'la' texto.txt* nos mostrará por pantalla aquellas líneas del fichero que contienen 'la'.

Fichero texto.txt
```
hola, que tal estas?
estoy en casa
de la familia Costa
saludos
```

```
hendrix02:$ egrep 'la' texto.txt
hola que tal estas?
de la familia Costa
hendrix02:$
```

Escribe en un fichero llamado *ej6.txt* cada una de las órdenes necesarias para mostrar por pantalla:

- las líneas de un fichero llamado *t1.txt* que empiecen y finalicen con un dígito;
- las lineas de *t1.txt* que contengan un número par de vocales;

[2]Las expresiones regulares son unas grandes amigas de los administradores de sistemas Linux. Todo lo que ocurre en una máquina Linux queda registrado en ficheros log, ficheros de texto donde se guarda información sobre distintos eventos: usuarios conectados, envío de mailes, direcciones IP de máquinas que intentan acceder al sistema, etc. Para filtrar esta información, las expresiones regulares son un herramienta básica que todo buen administrador debe dominar.

- las líneas de *t1.txt* que contengan un número impar en decimal. Por ejemplo, la línea *"hola 32dados"*, no se debe mostrar; sin embargo, *"hola 211dados"*, sí debe mostrarse.

Ejercicio 7

Existen diversos métodos para expresar partituras musicales mediante texto. Uno de los más comunes es la notación anglosajona donde cada nota de una escala viene dada por una letra entre la A y la G, de forma que nuestro tradicional do-re-mi-fa-sol-la-si (notación latina) es representado por C-D-E-F-G-A-B. Aunque las notaciones textuales contienen infinitud de símbolos para expresar sostenidos, bemoles, duraciones, pausas, etc, el formato que usaremos, y que se explica a continuación en este ejercicio, es una simplificación que básicamente nos permitiría tocar una melodía con las teclas blancas de un piano y un solo dedo.

La representación de las partituras mediante texto nos permite utilizar expresiones regulares para encontrar patrones que se repiten en una composición musical. Estos patrones son objeto de estudio por los musicólogos [3] y nos permiten resolver preguntas como cuál es patrón más común en los trastes de los solos de guitarra de Jimmy Hendrix o cuáles de los conciertos de Brandeburgo contienen el motivo B-A-C-H [4]?. Existen herramientas como *Humdrum* que nos facilitan la investigación musical permitiendo, entre otras funcionalidades, explorar las partituras utilizando expresiones regulares.

En el formato de fichero que utilizaremos cada nota viene representada por una letra de la A a la G y un número entre 1 y 8 que representa la octava. Por ejemplo, `C1D1E1F1G1A1B1` representaría tocar una escala del do al si en la primera octava. Llamaremos *escala* a la aparición de las notas de la C a la B en la misma octava, y las diferenciaremos de las que llamaremos *escalas asonantes* donde aparecen consecutivamente las notas de la C a la B, pero cada nota pudiendo ser de una octava distinta. Notad que una escala en particular es un ejemplo de escala asonante. Los caracteres permitidos en este tipo de fichero son las letras `[A-G]`, los números `[1-8]`, y el salto de línea para facilitar la lectura del fichero. Los saltos de línea solo podrán aparecer tras un número.

Se pide implementar en *Flex*, fichero *ej7.l*, para analizar un texto escrito bajo este formato (supondremos que el texto es correcto) y halle el número de veces que aparecen los siguientes efectos musicales:

- Número de escalas asonantes que aparecen en el fichero;
- número de veces que aparecen dos o más escalas asonantes consecutivas;

[3] Un ejemplo de artículo de investigación: `http://ismir2001.ismir.net/pdf/dovey.pdf`
[4] `https://es.wikipedia.org/wiki/Motivo_BACH`

- número de veces que aparecen dos o más escalas asonantes consecutivas siendo la primera escala de la serie en primera o segunda octava. Por ejemplo, si aparece C1D1E1F1G1A1B1C2D1E1F1G2A1B3 contabilizaremos una aparición de este efecto (también contabilizaremos dos escalas asonantes y una aparición de dos o más escalas asonantes consecutivas). Sin embargo, si aparece en el fichero C2D3E1F1G4A1B1C1D1E1F1G1A1B1, no se contabilizará (aunque también contaremos dos escalas asonantes y una aparición de dos o más escalas asonantes consecutivas).
- número de veces que aparecen consecutivamente C D E (do re mi) cada nota en cualquier octava.

La salida del programa que analiza un fichero constará de cuatro líneas de texto:

```
EA:<total escalas asonantes>
EMD:<total dos o más escalas asonantes consecutivas>
EPS:<total dos o más escalas asonantes consecutivas, primera
escala de la serie en primera o segunda octava>
DRM:<total apariciones do re mi cada nota en cualquier octava>
```

Ejemplo:

Entrada:

```
D2C1D1E1F1
G1A1B1
C4D1E8

F1G1A3B2
C1D2E3
```

Salida:

```
EA:2
EMD:1
EPS:1
DRM:3
```