Práctica 2
Manejo Avanzado de *Flex*

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

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:

- Verifica que todos tus ficheros fuente (*.l*) contienen como comentario en sus primeras líneas los NIPs 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 hendrix.
- Crea un fichero comprimido .zip llamado

  `nipPr2.zip`

  donde *nip* es el identificador personal de la persona que someta la práctica. 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

El objetivo 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

Diseña mediante Flex un programa que permita borrar todo tipo de comentarios de un fichero de código fuente sintácticamente correcto escrito en C++. El fichero de Flex se llamará **ej1.l**. Recordad que en C++ se pueden utilizar dos tipos diferentes de comentarios: los comentarios de línea que comienzan por `//` y terminan con la línea; y los comentarios de párrafo que se escriben entre `/*` y `*/`. Tened en cuenta que estas tres expresiones pueden aparecer dentro de una expresión de tipo texto (cadenas estilo "...") y esto no supone que comience o termine un comentario. Para simplificar el problema supondremos que dentro de una expresión de tipo texto no aparecerá la expresión \". Por ejemplo:

Entrada:

```
/* Comentario
 * de
 * parrafo
 */

#include <stdio.h>
#include "string"

const float PI = 3.1416;

// Metodo mostrar
void mostrar()
{
    printf( "Hola mundo /*/*/*/*\*\*\*\ \n" ); //cuidado
    printf( "mundo Hola \*\*\*\*/*/*/*/ \n" );
}
/* Metodo main */
void main()
{
    mostrar();
    if( PI > 2 * PI )
        printf( "Mala cosa \n" );
}
```

Salida:

```
#include <stdio.h>
#include "string"

const float PI = 3.1416;


void mostrar()
{
    printf( "Hola mundo /*/*/*/*\*\*\*\ \n" );
    printf( "mundo Hola \*\*\*\*/*/*/*/ \n" );
}

void main()
{
    mostrar();
    if( PI > 2 * PI )
        printf( "Mala cosa \n" );
}
```

Ejercicio 2

En la última década, el área de la Inteligencia Artificial ha sufrido una auténtica revolución con la aparición de modelos de redes neuronales profundas (deep-learning), llegando a ser habitual ver noticias como "Mona Lisa is brought to life by AI" donde se muestra cómo un algoritmo de deep-learning es capaz de realizar con un realismo asombroso una animación a partir de una imagen de la Mona Lisa.

La arquitectura de una red neuronal profunda indica qué capas la forman junto con una descripción de cada capa. La visualización de las diferentes arquitecturas cobra importancia a la hora de comprender su estructura, comparar la complejidad de los modelos, o analizar la corrección del número de parámetros establecido en las diferentes capas de la red. En la página `https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network` se pueden encontrar diferentes herramientas de visualización que pueden llegar a ser tan sorprendentes como GraphCore (número 17).

Nuestro interés en esta práctica se centra en keras-sequential-ascii que es una biblioteca que permite obtener la arquitectura de la red de forma textual e investigar la relación de sus parámetros en modelos sencillos. En la Figura 2.1 se puede encontrar una representación de la arquitectura VGG16-Net que ganó en 2014 el ImageNet challenge con una arquitectura realmente profunda para la época (`https://www.geeksforgeeks.org/vgg-16-cnn-model/`).

VGG 16 Architecture

```
      OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

          Input   #####      3  224  224
     InputLayer     |   -------------------         0     0.0%
                  #####      3  224  224
  Convolution2D    \|/  -------------------      1792     0.0%
           relu   #####     64  224  224
  Convolution2D    \|/  -------------------     36928     0.0%
           relu   #####     64  224  224
   MaxPooling2D   Y max -------------------         0     0.0%
                  #####     64  112  112
  Convolution2D    \|/  -------------------     73856     0.1%
           relu   #####    128  112  112
  Convolution2D    \|/  -------------------    147584     0.1%
           relu   #####    128  112  112
   MaxPooling2D   Y max -------------------         0     0.0%
                  #####    128   56   56
  Convolution2D    \|/  -------------------    295168     0.2%
           relu   #####    256   56   56
  Convolution2D    \|/  -------------------    590080     0.4%
           relu   #####    256   56   56
  Convolution2D    \|/  -------------------    590080     0.4%
           relu   #####    256   56   56
   MaxPooling2D   Y max -------------------         0     0.0%
                  #####    256   28   28
  Convolution2D    \|/  -------------------   1180160     0.9%
           relu   #####    512   28   28
  Convolution2D    \|/  -------------------   2359808     1.7%
           relu   #####    512   28   28
  Convolution2D    \|/  -------------------   2359808     1.7%
           relu   #####    512   28   28
   MaxPooling2D   Y max -------------------         0     0.0%
                  #####    512   14   14
  Convolution2D    \|/  -------------------   2359808     1.7%
           relu   #####    512   14   14
  Convolution2D    \|/  -------------------   2359808     1.7%
           relu   #####    512   14   14
  Convolution2D    \|/  -------------------   2359808     1.7%
           relu   #####    512   14   14
   MaxPooling2D   Y max -------------------         0     0.0%
                  #####    512    7    7
        Flatten   ||||| -------------------         0     0.0%
                  #####       25088
          Dense   XXXXX ------------------- 102764544    74.3%
           relu   #####        4096
          Dense   XXXXX -------------------  16781312    12.1%
           relu   #####        4096
          Dense   XXXXX -------------------   4097000     3.0%
        softmax   #####        1000
```

Figura 2.1: Arquitectura de VGG16-Net representada mediante keras-sequential-ascii.

La información proporcionada por keras-sequential-ascii para cada una de las capas de la red consiste dos líneas para cada capa con la siguiente estructura:

1. La identificación del tipo de capa (en el ejemplo, entrada, convolucional + relu,

maxpooling, flatten, densa, y softmax).

2. Separadores en cada una de las dos líneas, según el ejemplo que aparece en la figura 2.1 (y en los ficheros de texto de los ejemplos). Los caracteres no visibles son todos espacios, no hay tabuladores.

3. La dimensión de la entrada a dicha capa, en la primera línea, mediante números enteros no negativos separados por espacios. Notad que para algunas capas la dimensión es 3D (tres números bajo la columna DATA DIMENSIONS) mientras que para otras es 1D (un solo número bajo DATA DIMENSIONS). También podrían aparecer otras dimensiones nD, $n \geq 1$.

4. El tamaño de los pesos o parámetros de cada capa, en la segunda línea, mediante un entero no negativo.

5. Su porcentaje, en la segunda línea, mediante un número real con un decimal seguido del símbolo %.

Es posible procesar este fichero de manera automática para obtener información de interés de la arquitectura de la red. Para esta práctica se pide diseñar mediante Flex un programa que calcule la dimensión total de la red. Para ello, se utilizará una variable que vaya acumulando las dimensiones de cada una de las capas que nos encontremos en el fichero. Si la dimensión de la capa es mayor que uno, se multiplicarán entre si todas las componentes y se sumarán a nuestro acumulador. Si la dimensión de la capa es uno, simplemente se sumará el valor de la dimensión al acumulador. El fichero de Flex se llamará **ej2.l**. La salida tendrá el siguiente formato:

```
D: <dimension>
```

Adjunto al material de la práctica, están disponibles dos ficheros de entrada de ejemplo con formato keras-sequential-ascii, `VGG16.txt` y `CIFAR10.txt`. Las salidas correctas para estos dos ficheros son:

VGG16.txt
```
D: 15413224
```

CIFAR10.txt
```
D: 54506
```

Ejercicio 3

Con el uso de redes sociales como Twitter, donde los usuarios tienen unos pocos caracteres para poder enviar un mensaje, han aparecido servicios (bit.ly, TinyURL) que permiten acortar direcciones web y producir *alias* cortos de URLs más largas. Estas urls cortas se componen de:

`http://<nombreDominio>/<nombreRecurso>`

donde un `<nombreDominio>` es correcto si es una secuencia de letras, números y/o el carácter '.', y un `<nombreRecurso>` es correcto si es una secuencia de letras y/o números.

Una organización tiene contratado con bit.ly y TiniURL el siguiente servicio:

a) Los nombres de recursos bajo el dominio `bit.ly` serán cadenas formadas por un número múltiplo de tres de letras y un número impar de dígitos.

b) Los nombres de recursos bajo el dominio `tinyurl.com` serán cadenas formadas por un número que no sea múltiplo de tres de letras y un número par de dígitos.

El objetivo de la práctica es contar de un fichero cuántas URLs cortas tiene la organización con cada una de estas dos empresas, esto es, cuántas URLs cumplen la condición a) para el dominio `bit.ly`, y cuántas cumplen la condición b) para el dominio `tinyurl.com`. El formato del fichero de entrada será el siguiente:

```
http://<nombreDominio>
<nombreRecurso> <nombreRecurso>...
<nombreRecurso> <nombreRecurso>...
...
http://<nombreDominio>
<nombreRecurso> <nombreRecurso>...
<nombreRecurso> <nombreRecurso>...
...
```

Cada línea del fichero supondremos que obligatoriamente tiene uno de los siguientes formatos:

- una línea que comienza con `http://`, seguido de un `<nombreDominio>` correcto;
- una línea con nombres de recursos correctos separados por uno o varios espacios en blanco (la línea puede estar vacía);
- si en alguna línea aparece el carácter `#` se considerará que es el inicio de un comentario y no se tendrá en cuenta todo el texto entre `#` y el final de línea.

Siempre que aparece una línea de tipo `http://<nombreDominio>`, los siguientes nombres de recursos estarán asociados a dicho dominio hasta la aparición de una línea con otro nombre de dominio (o fin de fichero).

Implementar con Flex en un fichero llamado **ej3.l** un analizador lexicográfico que, dado como en entrada un fichero con el formato descrito anteriormente, tenga por salida dos líneas con el siguiente formato e información:

```
TB: <número de URLs de la organización asociadas a bit.ly>
TT: <número de URLs de la organización asociadas a tinyurl.com>
```

Ejemplo:

Entrada:

```
http://bit.ly
3R2 222

R5ll
http://otros.com
3R2 RRr1
http://bit.ly
3Rj2 #ejemplo 222
http://tinyurl.com
3R2 22
```

Salida:

```
TB: 1
TT: 1
```