

Grupo Jueves 12:00 – 14:00 semanas B
- Práctica 1 -
Autor : Rubén Rodríguez Esteban

Ejercicio 1:

1.1 Descripción del ejercicio:

Escribe un programa con Flex de nombre ej1.1 que sustituya cualquier correo de @hotmail por @gmail. Esto es, cada vez que aparezca la cadena @hotmail la cambie a @gmail.

1.2 Código del programa:

```
%%  
  
@hotmail { printf ( "%s" , "@gmail" ); }  
  
%%
```

El programa muestra un patrón definido entre los símbolos %% que tiene como función leer un conjunto de caracteres introducidos por el usuario, y si entre ellos figura la cadena @hotmail la sustituye imprimiendo por pantalla mediante la orden **printf** la cadena @gmail.

1.3 Conjunto de pruebas del funcionamiento del programa

A continuación, se muestran dos ejemplos del funcionamiento del programa en **hendrix**. El programa es ejecutado mediante el comando ./ej1. Observar en la imagen como la cadena @hotmail se ha sustituido correctamente por @gmail manteniendo invariante el resto de la información.

```
hendrix02:~/TeoComp/ ./ej1  
pedroPiicapiedra@hotmail.com ernesto@hotmail.com maria@hotmail.com  
pedroPiicapiedra@gmail.com ernesto@gmail.com maria@gmail.com
```

```
hendrix02:~/TeoComp/ ./ej1  
pabloruiz@hotmail.com sergiodominguez@hotmail.com  
pabloruiz@gmail.com sergiodominguez@gmail.com
```

Ejercicio 2:

2.1 Descripción del ejercicio:

Elabora un programa en Flex de nombre ej2.1 que permita contar el número de usuarios de correo de hotmail, esto es, el número de apariciones de la cadena @hotmail.

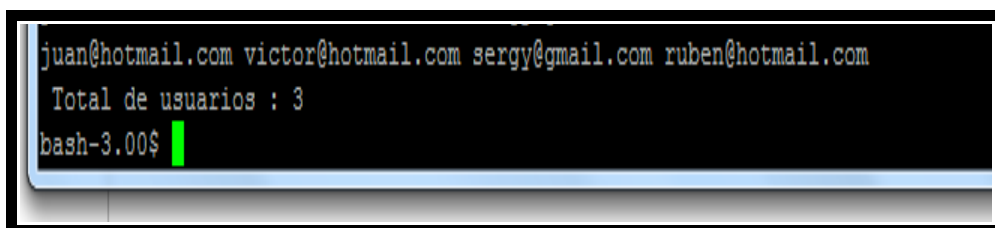
2.2 Código del programa

```
%{  
    int contador = 0;  
}%  
  
%%  
  
@hotmail { ++contador; printf ( "%s" , yytext); }  
  
%%  
  
int main(){  
    yylex();  
    printf ( " Total de usuarios : %d\n" , contador );  
}
```

En este programa entre los dos primeros símbolos %% he definido una variable de tipo entero llamada **contador** inicialmente a cero que va a guardar el número de usuarios de correo hotmail. El patrón definido a continuación incrementa la variable contador en uno cada vez que de los caracteres introducidos por el usuario detecta la cadena hotmail. Posteriormente, tengo una función **main()** que me devuelve por pantalla el valor de la variable contador.

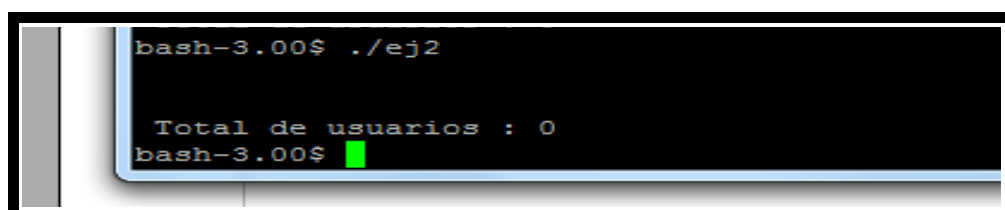
2.3 Conjunto de pruebas de funcionamiento del programa

El programa es ejecutado mediante el comando ./ej1. Observar en la imagen como el valor de contador muestra correctamente el número de usuarios hotmail, además el correo **sergy@gmail.com** no es contado dado que no contiene @hotmail.



```
juan@hotmail.com victor@hotmail.com sergy@gmail.com ruben@hotmail.com  
Total de usuarios : 3  
bash-3.00$
```

Resultado de la variable **contador** cuando no se introduce ningún correo.



```
bash-3.00$ ./ej2  
  
Total de usuarios : 0  
bash-3.00$
```

Ejercicio 3

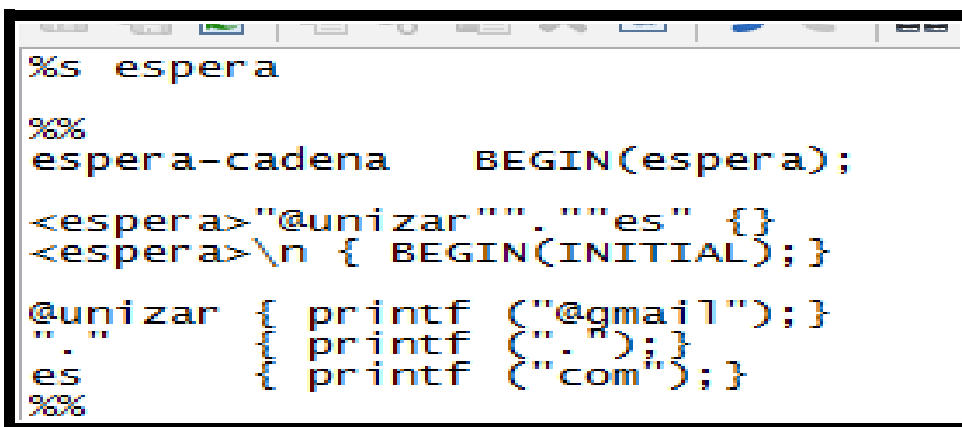
3.1 Descripción del ejercicio:

Escribe un programa con Flex de nombre ej3.1 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

Contestación razonada a las pregunta del ejercicio 3.

- Pregunta → 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 email? ¿Por qué?
- Respuesta → No lo escribirá correctamente puesto que el el programa lo interpretaría como tres tokens distintos: “jrg@unizaroes “, el punto “.” y el “es”. Para ello, hay que utilizar el comando <espera> seguido o bien de toda la expresión que buscamos indicando que debe hacer al encontrarla, es decir, por medio de un único patrón. Aunque también puede realizarse fragmentando la expresión en cada uno de los tokens anteriores indicando que hacer en cada caso, es decir, por medio de tres patrones distintos. Para este ejercicio, he utilizado el segundo mecanismo.

3.2 Código del programa

A screenshot of a text editor window showing the code for a Flex scanner. The code is as follows:

```
%s  espera
%%
espera-cadena  BEGIN(espera);
<espera>"@unizar"."es" {}
<espera>\n { BEGIN(INITIAL); }

@unizar { printf ("@gmail"); }
"." { printf ("."); }
es { printf ("com"); }
%%
```

3.3 Conjunto de pruebas de funcionamiento del programa

```
hendrix02:~/TeoComp/ ./ej3
pedro@unizar.es
pedro@gmail.com
fernando@unizar.es
fernando@gmail.com
ruperto@unizar.es
ruperto@gmail.com
hendrix02:~/TeoComp/
```

Ejercicio 4

4.1 Descripción del ejercicio:

Construye un programa en Flex de nombre ej4.1 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.

4.2 Código del programa

```
%%
[0-9]    { printf( "%d", atoi(yytext)+1);}
\n       { printf( "\n"); printf( "\n");}
%%
```

En este programa he creado dos patrones distintos:

- El primer patrón detecta el carácter de un número entre cero y nueve, obtiene mediante la función **atoi()** su valor en entero y después lo incrementa en uno imprimiéndolo por pantalla.
- El segundo patrón cada vez que detecta el carácter “ salto de línea “ realiza un salto de línea adicional

4.3 Conjunto de pruebas de funcionamiento del programa

```
bash-3.00$ ./ej4
Mi numero de telefono es el 659382321
Mi numero de telefono es el 7610493432

Mi madre nacio el 24 de octubre de 1975
Mi madre nacio el 35 de octubre de 21086

ponte en contacto con
ponte en contacto con

el asistente 59 en 04 minutos
el asistente 610 en 15 minutos

bash-3.00$ █
```

Observar que la función `atoi()` no una como parámetro todo el número, sino que coge como parámetro a cada carácter por separado y que al teclear el sato de línea el programa muestra la información introducida por el usuario seguida de un salto de línea adicional.

Ejercicio 5

5.1 Descripción del ejercicio:

Implementa un analizador léxico en Flex llamado `ej5.l` que analice un fichero de texto en formato csv, ficheros compuestos de líneas de texto formadas por grupos de valores separados por comas y genere otro con la siguiente información:

- cuántos caracteres tiene el fichero (excepto saltos de línea).
- cuántas líneas de texto;
- cuántos valores contiene el fichero;
- total de caracteres del valor de mayor longitud;
- total de valores que son numéricos (formados sólo por dígitos).

5.2 Código del programa

```

%{
int C = 0;
int L = 0;
int V = 0;
int M = 0;
int N = 0;
}%

%%
,          { V++;
            C += 1;
            }
[0-9]+     { N++;
            C += strlen(yytext);
            }
[0-9]*[a-zA-Z ]+[0-9]* { int parcial = strlen(yytext);
                        if (parcial > M){
                            M = parcial;
                        }
                        C += strlen(yytext);
                        }

\n        { L++;
            V++;
            }

%%

int main(){
    yylex();
    printf ( " Total de caracteres excepto saltos de línea --> C: %d\n" , C );
    printf ( " Total de líneas : %d\n" , L );
    printf ( " Total de valores : %d\n" , V );
    printf ( " Total de caracteres de valor mas largo : %d\n" , M );
    printf ( " Total de valores numericos : %d\n" , N );
}

```

Previamente se han definido las variables de tipo entero que van a llevar la cuenta de de los datos citados anteriormente todos ellos iniciados a cero

- C: caracteres del fichero sin contar los saltos de línea.
- L: número de línea que tiene el fichero.
- V: número de valores del fichero.
- M: valor con mayor longitud de caracteres.
- N: valores numéricos del fichero.

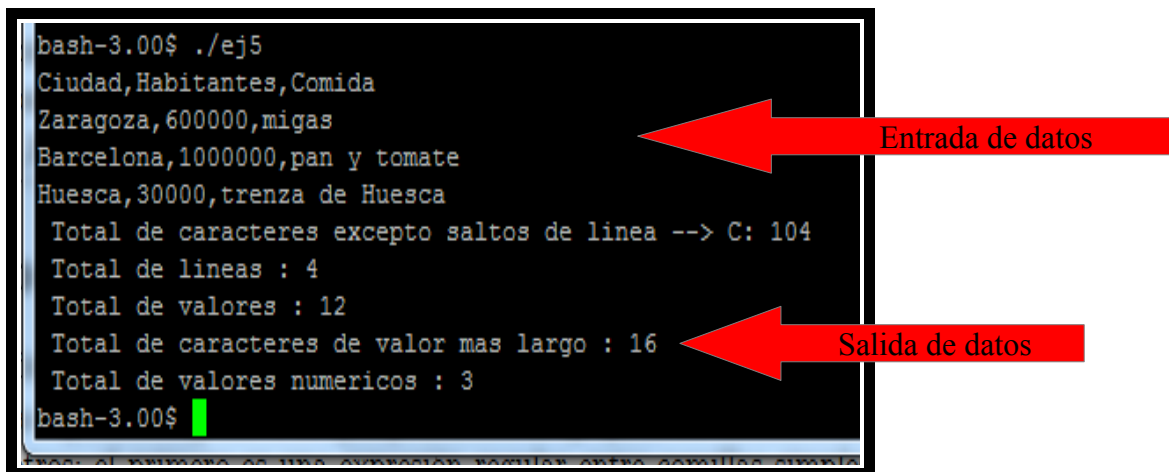
A continuación se expone una breve explicación de la tarea de los patrones definidos en el código anterior.

1. El primer patrón suma uno a los valores de V y C cada vez que lee el carácter “,”.
2. El segundo patrón suma uno al valor de N cuando lee cualquier combinación de caracteres numéricos e incrementa en valor de C en función del número de caracteres numéricos leídos mediante la función **strlen()**.
3. El tercer patrón cuenta la longitud de caracteres de un valor y lo compara con la longitud del valor anterior leído. Si dicha longitud es mayor que la anterior, se actualiza, en caso contrario, se deja como está.
4. El cuarto patrón lleva la cuenta de los saltos de línea del fichero, incrementando el valor de L y de V.

Por último se muestran por pantalla los resultados obtenidos

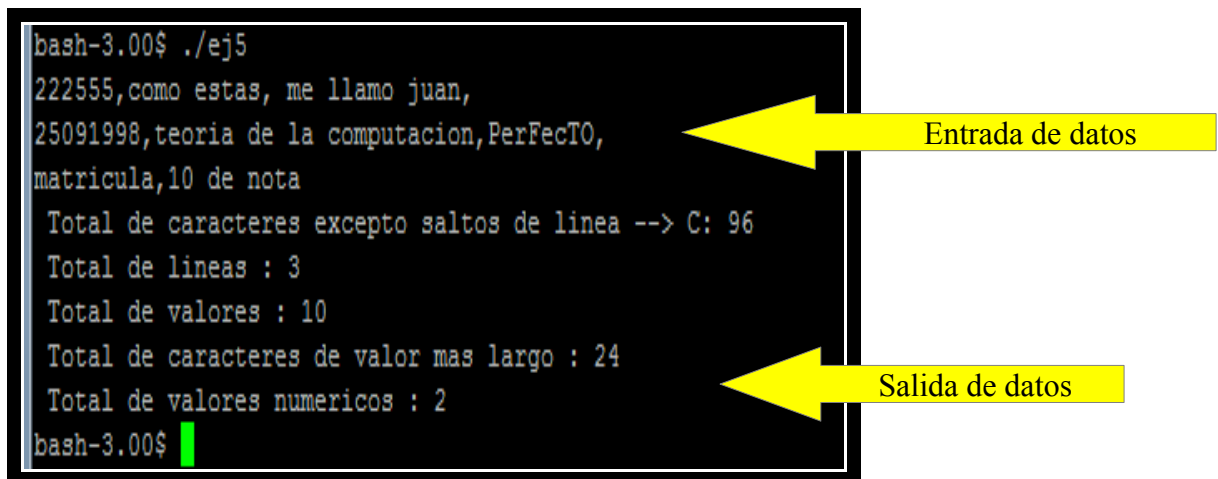
5.3 Conjunto de pruebas de funcionamiento del programa

```
bash-3.00$ ./ej5
Ciudad,Habitantes,Comida
Zaragoza,600000,migas
Barcelona,1000000,pan y tomate
Huesca,30000,trenza de Huesca
Total de caracteres excepto saltos de linea --> C: 104
Total de lineas : 4
Total de valores : 12
Total de caracteres de valor mas largo : 16
Total de valores numericos : 3
bash-3.00$
```



The screenshot shows the execution of the program ./ej5. The input consists of four lines of data: 'Ciudad,Habitantes,Comida', 'Zaragoza,600000,migas', 'Barcelona,1000000,pan y tomate', and 'Huesca,30000,trenza de Huesca'. The output consists of five lines: 'Total de caracteres excepto saltos de linea --> C: 104', 'Total de lineas : 4', 'Total de valores : 12', 'Total de caracteres de valor mas largo : 16', and 'Total de valores numericos : 3'. A red arrow points from the text 'Entrada de datos' to the input lines, and another red arrow points from the text 'Salida de datos' to the output lines.

```
bash-3.00$ ./ej5
222555,como estas, me llamo juan,
25091998,teoria de la computacion,PerFecT0,
matricula,10 de nota
Total de caracteres excepto saltos de linea --> C: 96
Total de lineas : 3
Total de valores : 10
Total de caracteres de valor mas largo : 24
Total de valores numericos : 2
bash-3.00$
```



The screenshot shows the execution of the program ./ej5. The input consists of three lines of data: '222555,como estas, me llamo juan,', '25091998,teoria de la computacion,PerFecT0,', and 'matricula,10 de nota'. The output consists of five lines: 'Total de caracteres excepto saltos de linea --> C: 96', 'Total de lineas : 3', 'Total de valores : 10', 'Total de caracteres de valor mas largo : 24', and 'Total de valores numericos : 2'. A yellow arrow points from the text 'Entrada de datos' to the input lines, and another yellow arrow points from the text 'Salida de datos' to the output lines.

Ejercicio 6:

6.1 Descripción del ejercicio

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 líneas de t1.txt que contengan un número par de vocales.
- las líneas de t1.txt que contengan un número impar en decimal.

6.2 Código del programa

A continuación se muestra el contenido del fichero ej6.txt con los patrones anteriores y una breve explicación:

En el fichero se muestran definidos los tres patrones buscados:

- El primer patrón detecta al menos uno o más caracteres numéricos seguido de cero o más caracteres no numérico seguido de al menos uno o más caracteres numéricos
- El segundo patrón busca cero o más carácter distinto de una vocal, seguido de una vocal, concatenado con cero más caracteres distintos de vocales (x2) ,dicha estructura se repite cero o más veces, seguida de cero o más caracteres distintos de vocal
- El tercer patrón concatena cualquier cifra de cero a nueve repetida cero o más veces seguida de un dígito impar.

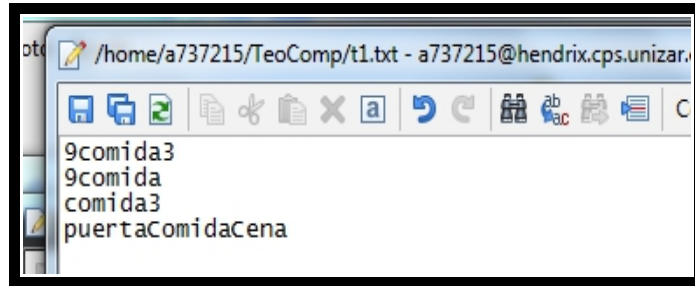
6.3 Conjunto de pruebas de funcionamiento del programa

A continuación se muestran las pruebas realizadas para asegurar la correcta ejecución de las órdenes anteriores. Para ello, se muestra el contenido del fichero t1.txt y la salida por pantalla en **hendrix** de los resultados en cada una

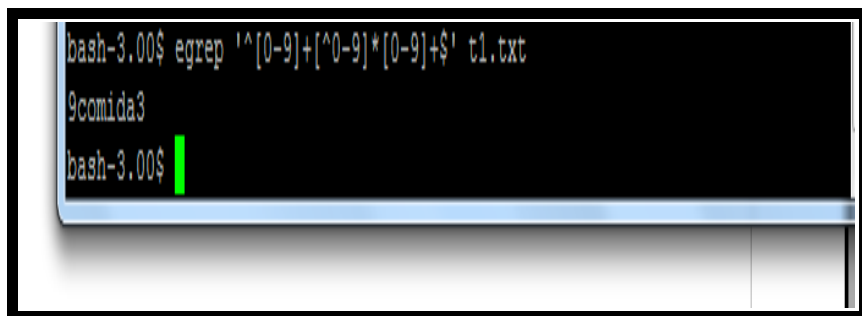
de las situaciones

- Primera situación

Contenido inicial del fichero t1.txt

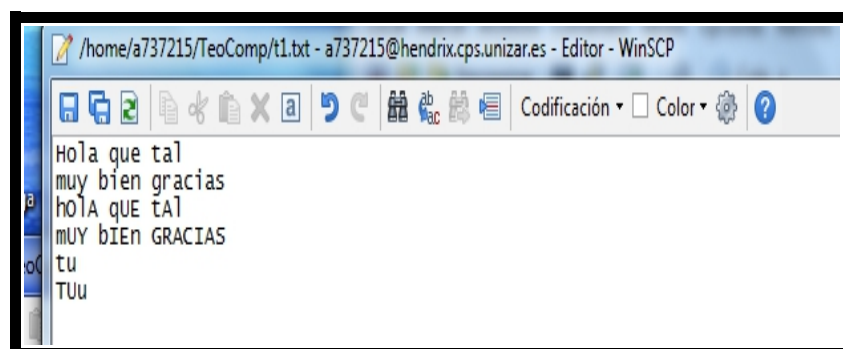


Resultados obtenidos en hendrix al compilar el programa



- Segunda situación

Contenido inicial del fichero t1.txt

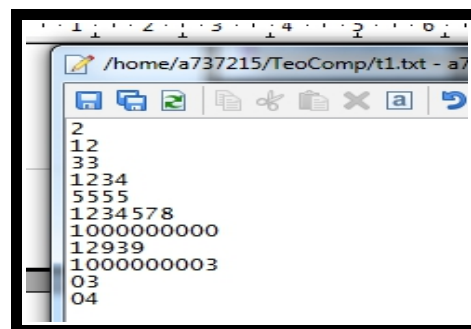


Resultados de la ejecución en **hendrix** del programa

```
bash-3.00$ egrep '^[^aeiouAEIOU]*([aeiouAEIOU][^aeiouAEIOU]*[aeiouAEIOU]) *$' t1.txt
muy bien gracias
mUY bIEEn GRACIAS
TUu
```

- Tercera situación

Contenido inicial del fichero t1.txt



```
2
12
33
1234
5555
1234578
1000000000
12939
1000000003
03
04
```

Resultados obtenidos en hendrix al compilar el programa

```
bash-3.00$ egrep '^[0-9]*[13579]+$' t1.txt
33
5555
12939
1000000003
03
bash-3.00$
```

Ejercicio 7

7.1 Descripción del ejercicio:

Implementa en Flex un programa de nombre ej7.1 que detecte las apariciones de números decimales que no estén en binario y sean múltiplos de 4, y añada a continuación del número (MULTIPLO4). Un número estará en binario si sólo se compone de 0 y 1.

7.2 Código del programa

```
%%  
(0|1)* {printf ( "%s" , yytext);}   
[048]|([0-9]*([02468][048]|[13579][26])) { printf(yytext);  
                                           printf("(MULTIPL04)");}  
%%
```

He definido dos patrones:

1. El primer patrón cada vez que detecta una cadena formada solo por 0 y 1 reescribe dicha cadena sin modificarla
2. El segundo patrón detecta cualquier número múltiplo de 4. De tal forma que cuando los detecta lo reescribe seguido del mensaje (MULTIPL04)

7.3 Reflexión acerca del ejercicio

En este ejercicio es de vital importancia para su correcto funcionamiento el orden en el que se definen los patrones ya que si el usuario teclea la cadena 100 y el patrón detector de múltiplos de 4 hubiese sido el primero en definirse lo detectaría como múltiplo cuando no lo es porque esta en binario. Por ello, no funcionaría adecuadamente el programa

7.4 Conjunto de pruebas de funcionamiento del programa

```
hendrix02:~/TeoComp/ ./ej7  
Esto es el numero 210  
Esto es el numero 210  
100 esta escrito en binario y es equivalente a 8 en decimal  
100 esta escrito en binario y es equivalente a 8(MULTIPL04) en decimal  
44 4096 312 son enteros numerables y multiplos de 4 pero no de 5  
44(MULTIPL04) 4096(MULTIPL04) 312(MULTIPL04) son enteros numerables y multiplos  
de 4(MULTIPL04) pero no de 5  
1001 001 101 1111001 estan en binario  
1001 001 101 1111001 estan en binario
```