Grupo Jueves 12:00 – 14:00 semanas B
- Práctica 2 -
Autor : Rubén Rodríguez Esteban

Ejercicio 1:

- Descripción del ejercicio:

  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. En el fichero adjunto a la práctica virussignatures.txt Aparecen unas 1.300 firmas de distintos virus conocidos. Este fichero se compone de líneas de texto con el siguiente formato:

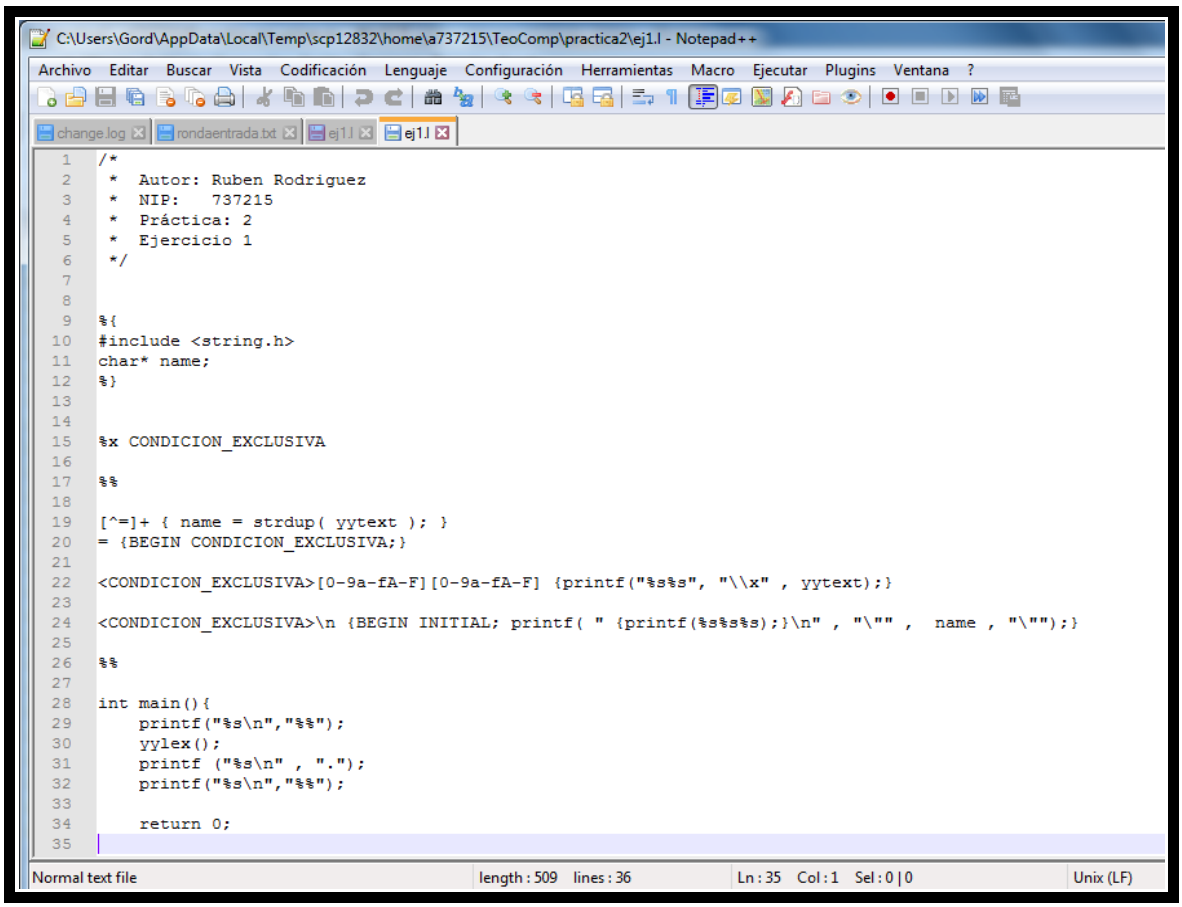
  1. nombre del virus: compuesto por cualquier carácter,excepto el carácter '=';
  2. el carácter =';
  3. la firma del virus expresada con dígitos en hexadecimal: la firma está compuesta por los caracteres del 0 al 9 y las letras de la A a la F (tanto en mayúsculas como en minúsculas). Cada dos dígitos en hexadecimal consecutivos representan un byte de la firma del virus. Por ejemplo, si en el fichero apareciese 32 A 426, significa que la firma del virus está compuesta por tres bytes cuyos códigos ascii en hexadecimal son 32,A4 y 26, respectivamente.

  El proceso para generar el programa que detecte las firmas contenidas en el fichero virussignatures.txt será el siguiente:

  1. 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.
  2. Se generará un fichero de Flex (vdetect.l) con tantas reglas como firmas de virus aparecen en el fichero. El patrón de cada una de estas reglas será la secuencia de bytes de la firma del virus, y la acción asociada será simplemente mostrar por pantalla el nombre del virus.
  3. Una vez creado automáticamente este fichero de Flex, se puede compilar 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.

- Código del programa

  A continuación, se muestra el código del fichero ej1.l



Este fichero genera un fichero que va a ser utilizado para poder realizar los pasos siguientes del ejercicio. Este fichero al ser invocado genera un nuevo fichero al que he llamado de acuerdo con el guión vdetec.l y su formato es exactamente el mismo que el del fichero virussignatures.txt aunque, la firma del virus ha sido separada por cada dos caracteres por un "\x" para indicar que está escrito en lenguaje hexadecimal. En este fichero he empleado un patrón que lee toda la información de un virus salvo el carácter "=" y que cuando es detectado por medio de la condición exclusiva escriba la firma del virus con la especificación descrita anteriormente.

A continuación, muestro como compilar el programa ej1.l en hendrix y como generar el nuevo fichero vdetect.l

```
bash-3.00$ flex ej1.l
bash-3.00$ gcc lex.yy.c -lfl -o ej1
bash-3.00$ ./ej1 <virussignatures.txt>
bash-3.00$
```

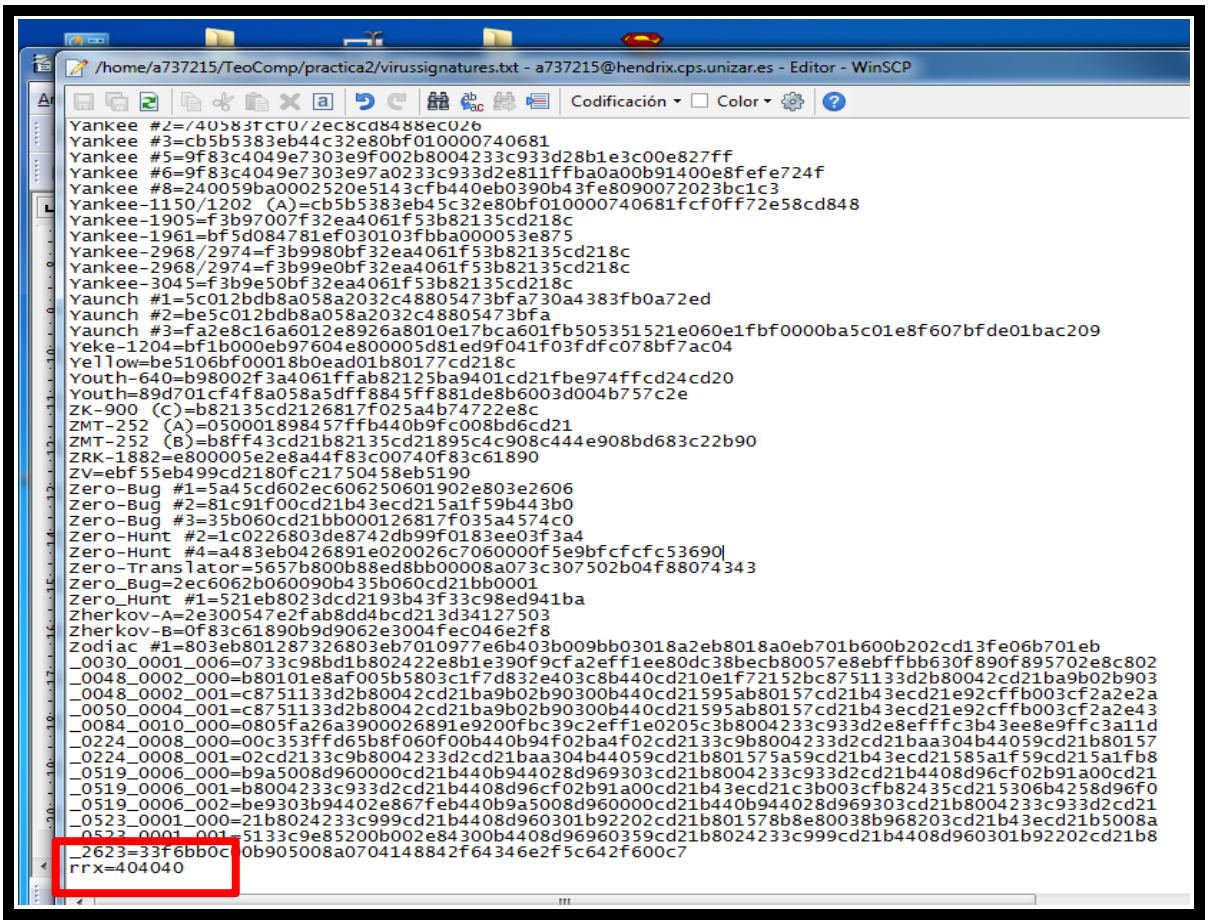
Una vez generado el fichero vdetect.l, procedemos a compilarlo en hendrix añadiendo la instrucción -ca para evitar errores como consecuencia de ser un fichero muy largo

```
hendrix02:~/TeoComp/practica2/ bash
bash-3.00$ flex -Ca vdetect.l
bash-3.00$ gcc lex.yy.c -lfl -o vdetect.l
bash-3.00$ ./vdetect.l <mal.vir> salida.txt
bash-3.00$
```

- Conjunto de pruebas de funcionamiento del programa

  Para facilitar la comprobación del programa he modificado un poco el fichero inicial creando un virus propio de nombre rrx y de signatura 404040 que se corresponde con el carácter @@@ puesto que la @ tiene como código ascii el 40 en hexadecimal.

  Contenido del fichero signatures.txt

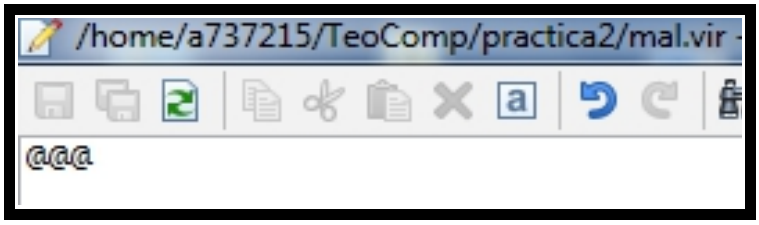


En el recuadro rojo de la imagen aparece el virus especificado anteriormente

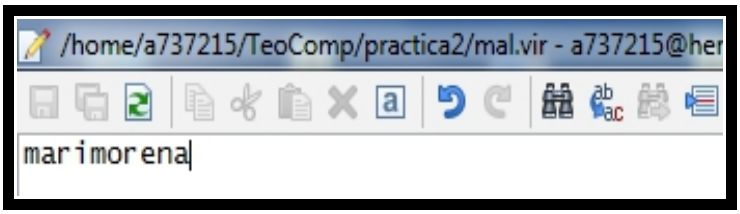
Ahora muestro el contenido del fichero que voy a analizar para ver si está o no infectado. Para ello, vamos a estudiar dos casos: un primer caso, llamado A, en el que el fichero está infectado, y un segundo caso llamado B, en el que el fichero no está infectado. En ambos casos vamos a estudiar el contenido del fichero salida.txt y ver si se obtienen los resultados correctos.

Contenido del fichero mal.vir

- Caso A



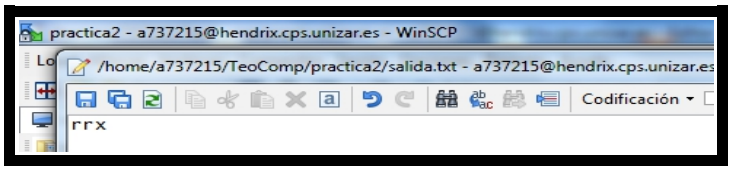
- Caso B



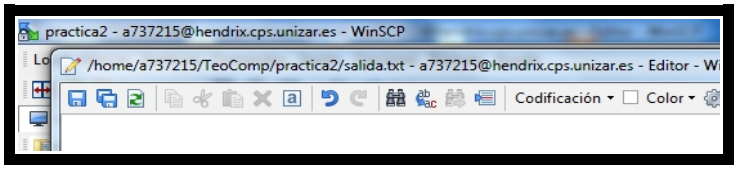
A continuación, plasmo el resultado final del programa en el fichero salida.txt ejecutando los comandos vistos anteriormente en hendrix. El fichero de salida devuelve los siguientes resultados:

Contenido del fichero salida.txt

- Caso A



- Caso B



Se puede comprobar que en el caso A, el programa a almacenado el nombre del virus por el que está infectado el fichero, concretamente el rrx. Sin embargo, no había infección en el caso B, y por esa razón el fichero de salida está en blanco.

EJERCICIO 2

- Descripción del ejercicio

  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++.

  El programa mostrará por pantalla el valor de los respectivos contadores y la correspondiente clasificación al lenguaje de programación correspondiente: C++ o Java.

- Código del programa

/home/a737215/TeoComp/practica2/ej2.l - a737215@hendrix.cps.unizar.es - Editor - WinSCP

Codificación ▾ Color ▾

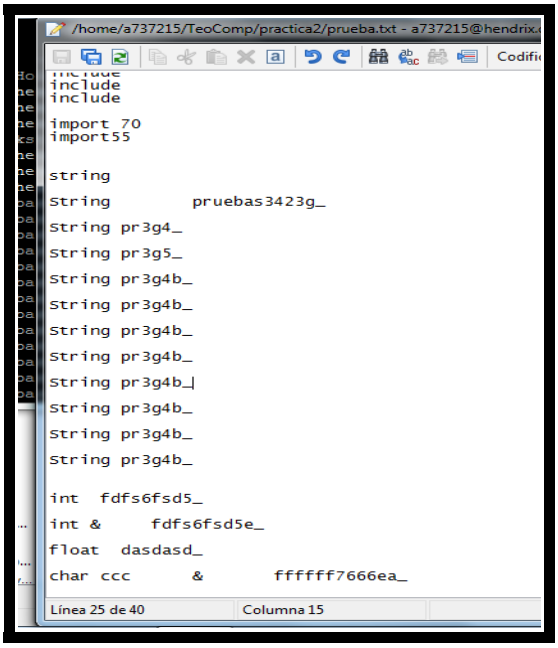
```
/*
 *
 * Autor: Ruben Rodriguez
 *  NIP:   737215
 *  Práctica: 2
 *  Ejercicio 2
 *
 */

%x COMENTARIO_LARGO
%x COMENTARIO_CORTO
%x CADENA

%{
#include <stdbool.h>
int contadorJ=0;
int contadorC=0;
int vecesJ = 0;
int vecesC = 0;
bool includeDetectado=false;
bool importDetectado=false;
%}

%%
"/*"     { BEGIN (COMENTARIO_LARGO);}
"//"     { BEGIN (COMENTARIO_CORTO);}
"\""        { BEGIN (CADENA);}
^[ ]*(include) { if (!includeDetectado){
                                        contadorC=contadorC+10;
                                        includeDetectado = true;
                                }}
^[ ]*(import)           { if (!importDetectado){
                                        contadorJ=contadorJ+10;
                                        importDetectado = true;
                                }}
[String][ ]*[a-zA-Z0-9]+[^0-9][_]  {  if ( vecesJ < 5){
                                                        contadorJ=contadorJ+1;
                                                        vecesJ=vecesJ+1;
                                                }
                                        }
[int|char|float][ ]*[\&][ ]*[a-zA-Z0-9]+[^0-9][_] {  if ( vecesC < 5){
                                                        contadorC=contadorC+1;
                                                        vecesC=vecesC+1;
                                                }
                                        }
<COMENTARIO_LARGO>"*/" { BEGIN (INITIAL);}
<COMENTARIO_LARGO>.|\n
<COMENTARIO_CORTO>\n { BEGIN (INITIAL);}
<COMENTARIO_CORTO>.|\n
<CADENA>"\"" { BEGIN (INITIAL);}
<CADENA>.|\n
.
%%
```

Línea 55 de 67 | Columna 1 | Codificando: 1252 (ANSI -

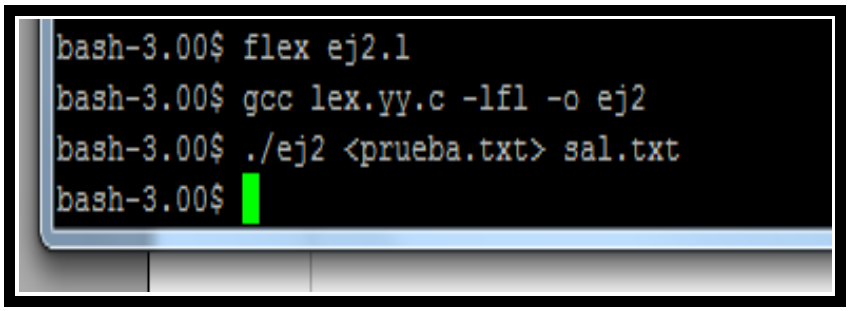
```
void main(){
        yylex();
        printf (" PJ : %d\n" , contadorJ );
        printf (" PC : %d\n" , contadorC );
        printf (" CLASIFICADO : ");
        if ( contadorC > contadorJ ){
                printf (" uno de C++\n");
        }
        else{
                printf (" uno de JAVA\n");
        }
}
```

- Conjunto de pruebas de funcionamiento del programa

  Código del programa de ejemplo: prueba.txt



  A continuación, se muestra como compilar en hendrix el ejercicio y donde mirar los resultados del programación.



  Al ejecutar el programa, los resultados se guardan en un fichero de texto llamado sal.txt

  Contenido del fichero sal.txt