Grupo Jueves 12:00 – 14:00 semanas A

- Práctica 2 -

Autor: Inés Román Gracia

NIP: 820731

EJERCICIO 1

1. Resumen

Para borrar los comentarios de un programa en lenguaje c he utilizado tres estados excluyentes entre sí: *comentario* para borrar los comentarios de varias líneas que borra todo lo que hay entre las expresiones */** y ***/*, *simples* que borra los caracteres de una línea después de la expresión *//* y, por último, *comillas* para no borrar nada que vaya entre comillas, aunque entre las comillas estén las expresiones */**, ***/* o *//*.

Código usado para borrar los comentarios de varias líneas:

```
"/*"      BEGIN(comentario);
<comentario>[^*\n]*
<comentario>"*"+[^*/\n]*
<comentario>\n
<comentario>"*"+"/" BEGIN(INITIAL);
```

Como se puede ver en este fragmento de código, el estado se inicia cuando aparece la expresión */**, después borramos todos los caracteres de cada línea a excepción de los asteriscos y también los asteriscos que no vayan seguidos de */*, ya que esto indica el final del comentario. Borramos también los saltos de línea y, por último, terminamos de borrar el comentario cuando encontremos la expresión ***/* o con más asteriscos.

Código usado para borrar los comentarios simples:

```
"//" {BEGIN(simples);}
<simples>.
<simples>\n {printf("\n"); BEGIN(INITIAL);}
```

Empezamos a borrar cuando encontramos la expresión *//*, borramos todos los caracteres a excepción de los saltos de línea, ya que cuando encontremos uno habremos acabado de borrar el comentario.

Código usado para no borrar nada entre comillas:

```
["]       {printf("%s", yytext); BEGIN(comillas);}
<comillas>["]    {printf("%s", yytext); BEGIN(INITIAL);}
```

No borra nada que vaya entre comillas y vuelve a escribir las comillas en el fichero que lee.

2. Pruebas

Los ficheros que he utilizado para comprobar el buen funcionamiento del programa son *ficheroc.txt* de entrada y *sficheroc.txt* de salida.

EJERICIO 2

1. Resumen

Para este ejercicio sólo he necesitado un estado para leer los números de cada línea que contenga dimensiones y así calcular la dimensión total.

Código empleado para leer las dimensiones de cada capa y calcular la dimensión total:

```
#####   {BEGIN(linea); capa = 1;}

<linea>[0-9]+ {capa = atoi(yytext)*capa;}

<linea>.

<linea>\n   {BEGIN(INITIAL); dimension += capa;}

\n

.
```

Todas las líneas que contienen dimensiones contienen antes la expresión *#####*, por tanto, el estado empezará con esta expresión y terminará al leer el salto de línea.

Al empezar a leer las dimensiones de cada capa inicializo el valor de la capa a 1 ya que es el elemento neutro de la multiplicación, y así, el resultado final de *capa* al terminar de leer la línea será la multiplicación de las dimensiones de esa capa. Sumo el valor de la capa al valor de la dimensión, inicializado al principio del programa a 0, para obtener la dimensión total como la suma de las dimensiones de todas las capas.

Por último borro todos los caracteres y los saltos de línea para que en el fichero de salida solo haya una línea con el resultado final de la dimensión.

2. Pruebas

Para comprobar este ejercicio he probado con los ficheros *CIFAR10.txt* y *VGG16.txt*, aportados en los materiales de la práctica, y de salida s*CIFAR10.txt* y s*VGG16.txt*, respectivamente. Se puede ver que el resultado es correcto.

EJERCICIO 3

1. Resumen

Para este ejercicio he utilizado tres estados. El primero identifica el comienzo de un dominio de protocolo http al leer la expresión *http://*. Los otros dos estados aparecen a partir del primero cuando se trata de un dominio *bit.ly* o *tinyurl.com* y terminan cuando leen de nuevo *http://* que los lleva al primer estado.

Además, con esta expresión ignoramos los comentarios en cualquiera de estos tres estados:

```
<*>"#".*\n
```

Código usado para comprobar si los recursos de *bit.ly* son correctos:

```
"http://"     BEGIN(dominio);
<dominio>"bit.ly"     BEGIN(bit);
<bit>[A-Za-z0-9]* {
   caracter = 0;
   digito = 0;
   for(int i = 0; i < yyleng; i++){
      if(isalpha(yytext[i])){
         caracter++;
      }
      else if(isalnum(yytext[i])){
         digito++;
      }
   }
   if(caracter % 3 == 0 && digito % 2 != 0){
      b++;
   }
}
<bit>"http://"  BEGIN(dominio);
```

Leo cada una de las expresiones que podrían ser recursos, es decir, conjuntos de números y letras y después mediante las funciones *isalpha* e *isalnum* cuento el número de caracteres y de dígitos. Finalmente, si los caracteres son múltiplo de 3 y el número de dígitos es impar aumento el contador de recursos correctos de *bit.ly*.

De forma parecida compruebo los recursos de *tinyurl.com* que serán correctos cuando el número de letras no sea múltiplo de 3 y el número de dígitos sea par:

```
<dominio>"tinyurl.com"    BEGIN(tinyurl);
<tinyurl>[A-Za-z0-9]* {
   caracter = 0;
   digito = 0;
   for(int i = 0; i < yyleng; i++){
      if(isalpha(yytext[i])){
         caracter++;
      }
      else if(isalnum(yytext[i])){
         digito++;
      }
   }
   if(caracter % 3 != 0 && digito % 2 == 0){
      t++;
   }
}
<tinyurl>"http://"  BEGIN(dominio);
```

Por último, borro para todos los estados los caracteres restantes y los saltos de líneas para que en el fichero de salida solo este el resultado final del número de recursos correctos de *bit.ly* y de *tinyurl.com*:

```
<*>.
<*>\n
```

2. Pruebas

Los ficheros que he utilizado para comprobar el resultado de este ejercicio son *dominios.txt* de entrada y *sdominios.txt* de salida.