Grupo Miércoles 17:00 - 19:00 semanas A

–Práctica 1–

Autor: Razvan Ghita Calangiu
NIP: 927460

Autor: Rodrigo Herce Alonso
NIP: 935413

Ejercicio 1

1. Resumen

Para este ejercicio lo primero que hemos hecho ha sido declarar en la primera parte del código las variables: c (Caracter), l (Líneas), lv (Líneas vacías), v (Valores), m (Carácter de valor más largo), d (Duplas numéricas), n (Valores sin dígito) y las hemos iniciado a 0 para usarlas como contadores.

Por el orden de prioridad lo primero que hemos detectado ha sido los /n o salto de línea donde al detectarlo se incremente en 1 la variable l.

En segundo lugar de prioridad hemos detectado las duplas numéricas poniendo la expresión regular [0-9]*[:][0-9]* donde incrementamos en uno d y v y c en el tamaño de yyleng.

En tercer lugar de prioridad lo que hemos hecho ha sido detectar las cadenas de caracteres sin dígitos ya que si lo ponemos después de con dígitos por el orden de prioridad esto no se llega a ejecutar nunca además de los espacios y comas que si no partirían las cadenas más largas. La expresión regular es ([a-zA-Z:][a-zA-Z:]*[a-zA-Z:])|[a-zA-Z:] donde en la primera parte lo que hacemos es poner las cadenas de más de un carácter haciendo que empiecen por algo distinto de espacio luego ponemos la posibilidad de que haya espacios y por último nos aseguramos que no finalice en espacio, en la segunda parte añadimos las cadenas de 1 carácter. Con todo esto incrementamos en 1 la variable v y n y c en yyleng.

En cuarto lugar de prioridad lo que hemos hecho ha sido detectar las cadenas de caracteres con dígitos ya que es más importante que los espacios y las comas y menos importante que el anterior ya que sino no podríamos separar el caso de sin dígitos. La expresión regular es: ([a-zA-Z:0-9][a-zA-Z:0-9]*[a-zA-Z:0-9])|[a-zA-Z:0-9] este funciona muy parecido al anterior lo único que cambia es la inclusión de números en ambas posibilidades. Con esto incrementamos la variable v en uno y c en yyleng.

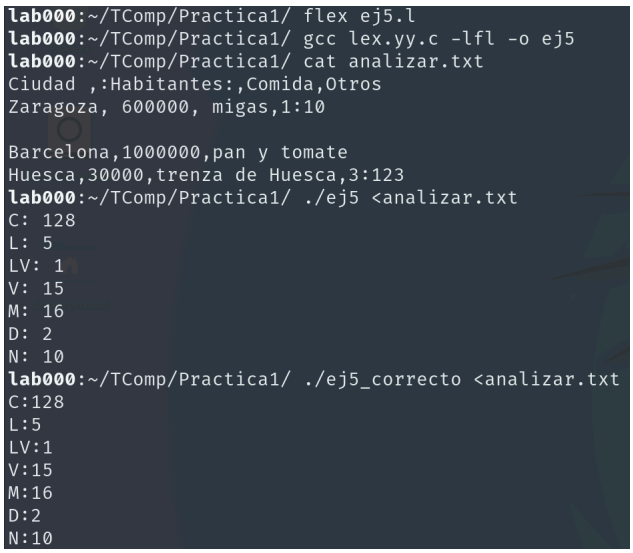
Estos tres últimos casos también comparar el valor de m (que sería la cadena más larga hasta el momento) con yyleng en caso de que yyleng sea mayor que m ma pasa a ser yyleng.

En quinto lugar lo que hemos hecho ha sido detectar las líneas con solo espacios a través de la expresión regular: ^[ ]*\n que aquí con el pico obliga a que empiece la línea por espacios y solo puede ser una sucesión de espacios seguidos por /n. Esto incrementa en 1 tanto l como lv y c lo incrementa en yyleng.

Por último para todos los posibles casos de caracteres no contemplado en lo anterior hemos puesto un . solo ya que si le ponemos la estrella de kleene como flex pilla la cadenas más larga terminaría cogiendo todo el texto. Y esto simplemente incremente c en yyleng que es 1.

2. Pruebas

Hemos hecho pruebas con analizar.txt tanto con nuestro código como con el correcto.



Ejercicio 2

1. Resumen

En este ejercicio en vez de usar las expresiones regulares en flex hacemos uso de Egrep donde definimos:

egrep '^[^0-9].*[0-9]$' t.txt la cual nos da todos los conjuntos cuya línea comienza con algo distinto de un número luego tiene cualquier carácter y finaliza la línea con un número.

egrep '^[ ]*[a-Z][a-Z0-9]*[ ] + [0-9] [0-9] ? [0-9] ?\. [0-9] [0-9] ? [0-9] ?\. [0-9] [0-9] ? [0-9] ?\. [0-9][0-9]?[0-9]?[ ]+[a-Z][a-Z0-9]*[ ]*' t.txt (he puesto espacios por el formato en word) En esta expresión lo que tenemos es el inicio de la línea con la posibilidad de espacio/os en blanco luego una letra seguido de 0 o más letras o números seguido de 1 o más espacios luego define una dirección ip seguido de uno o más espacios y por ultimo seguido de una letra con la posibilidad de alargarla a más caracteres o cifras finalizando con 0 o más espacios.

egrep '.*[1+3+5+7+9][^0-9].*|.*[1+3+5+7+9]$' t.txt En esta expresión inicia la cadena con cualquier cadena de caracteres seguida de un número impar seguido de algo que no sea un número y luego cualquier cadena o sino cualquier cadena seguida de un número impar y luego termina la línea. Es decir que en la línea haya un número impar.

2. Pruebas

```
  fermin37  25.255.21.4
  fermin37  25.255.21.4 pepe
  37fermin  25.255.21.4 pepe
  fermin37  25.255.21
  fermin37  25.255.21.8.67
  fermin37  25.255..21.8
  fermin37  25.2556.1.8
  fermin37  25.256.1.8 32fermin
  hola22 22dados
  hola 24mundo
  132hola 212dados
  131hola 212dados
  22 23
  12345
  5432
3
```

```
Autor 1: Razvan Ghita Calangiu 927460 Autor 2: Rodrigo Herce Alonso 935413 Practica 1 Ejercicio 6
egrep '^[^0-9].*[0-9]$' t.txt
egrep '^[ ]*[a-Z][a-Z0-9]*[ ]+[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?[ ]+[a-Z][a-Z0-9]*[ ]*' t.txt
egrep '.*[1+3+5+7+9][^0-9].*|.*[1+3+5+7+9]$' t.txt
```

```
lab000:~/TComp/Practica1/ egrep '^[^0-9].*[0-9]$' t.txt
  22 23
  5432
```

```
lab000:~/TComp/Practica1/ egrep '^[ ]*[a-Z][a-Z0-9]*[ ]+[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?\.[0-9][0-9]?[0-9]?[ ]+[a-Z][a-Z0-9]*[ ]*' t.txt
  fermin37  25.255.21.4 pepe
```

```
lab000:~/TComp/Practica1/ egrep '.*[1+3+5+7+9][^0-9].*|.*[1+3+5+7+9]$' t.txt
  fermin37  25.255.21.4
  fermin37  25.255.21.4 pepe
  37fermin  25.255.21.4 pepe
  fermin37  25.255.21
  fermin37  25.255.21.8.67
  fermin37  25.255..21.8
  fermin37  25.2556.1.8
  fermin37  25.256.1.8 32fermin
  131hola 212dados
  22 23
  12345
3
```

Ejercicio 3

1. Resumen

En el ejercicio 3 hemos vuelto a Flex lo único que esta vez no se podían usar contadores. Para ello hemos tenido que hacer un uso más preciso y complejo de las expresiones regulares.

De vuelta a el orden de prioridad lo más importante es la expresión regular: B(R*UR*UR*UR*)*E que lo que hace es al ver el B (inicio de la cadena buscada) contar que haya 3 U intercaladas por 0 o más R entre ellas y esto repetido 0 o más veces por lo que se busca es un número múltiplo de 3 de U. Cuando encuentra una cadena con estas características pone ++ al inicio de la cadena y al final.

En el segundo nivel de prioridad la expresión que buscamos es: B(R*UR*U)*R*UR*E donde lo que se hace es ver la primera B luego hay 2 U seguidas por 0 o más R esto 0 o más veces por lo que consigues un número par de U posteriormente lo que hace es poner otra U por lo que este número par se vuelve impar. En caso de que el número impar sea un número múltiplo de 3 no se ejecuta por el orden de prioridad. Esto luego imprime un – delante y detrás de la cadena.

En el último caso de prioridad la expresión que buscamos con flex es B(R*UR*)*R*UR*E los de lo que se hace es poner un número x de U y R intercaladas y posteriormente añade otra U. De primeras puedes pensar que puede dar lugar a un número impar de U pero como este caso se quedaría por orden de prioridad en el anterior y los múltiplos de 3 en el primero entonces lo único que nos queda son un número de U pares no múltiplos de 3. Al llegar a la E termina la cadena pone :: al principio y al final de la cadena.

2. Pruebas

```
lab000:~/TComp/Practica1/ flex ej7.l
lab000:~/TComp/Practica1/ gcc lex.yy.c -lfl -o ej7
lab000:~/TComp/Practica1/ cat fran.txt
querido francisco:
me puedes marcar como hemos quedado
las cadenas
BRRRRUE, BRUURUUE,
BUURURRRUUURRRRRE
BUUUUUUUUUE BRUE
lab000:~/TComp/Practica1/ ./ej7 <fran.txt
querido francisco:
me puedes marcar como hemos quedado
las cadenas
-BRRRRUE-, :BRUURUUE:,
++BUURURRRUUURRRRRE++
++BUUUUUUUUUE++ -BRUE-
lab000:~/TComp/Practica1/ ./ej7_correcto <fran.txt
querido francisco:
me puedes marcar como hemos quedado
las cadenas
-BRRRRUE-, :BRUURUUE:,
++BUURURRRUUURRRRRE++
++BUUUUUUUUUE++ -BRUE-
```

```
querido francisco:
me puedes marcar como hemos quedado
las cadenas
BRE, BRUURUUE,
BUURURRRUUURRRRRE
BUUUUUUUUUE BRUE
```