Práctica 2:
Expresiones regulares con `flex++`

Modelos de Computación

Grado en Ingeniería Informática
Curso 2019/20

UNIVERSIDAD
DE GRANADA

ALONSO BUENO HERRERO
MANUEL CASTELLÓN REGUERO
PABLO LOMBARDERO ROS

Índice

1 Introducción

En el presente documento exponemos el trabajo realizado sobre el tratamiento de expresiones regulares con la herramienta `flex++`. La idea básica es procesar un fichero de entrada donde identificar correos electrónicos y sus respectivas contraseñas, para después hacer un procesamiento básico de los mismos, ordenando cada tupla {`correo,contraseña`} por orden alfabético según el nombre de dominio del *e-mail* (es decir, que el par asociado a un correo con dominio `ugr.es` iría después que uno que fuese `atc.es`, por ejemplo).

Para realizar esta tarea se van a usar tres expresiones regulares, que identifican:

1. a la palabra asociada al correo electrónico: se trata de una cadena del tipo `<usuario>@<dominio>`;
2. a un elemento separador entre el correo y la contraseña que los identifica en el fichero de entrada. En concreto, los separadores serán el *espacio* en blanco, el *salto de línea* y el *tabulador*, combinados entre ellos como se quiera.
3. la contraseña, que puede contener caracteres alfabéticos, en mayúscula y/o minúscula, dígitos numéricos (del 0 al 9) y algunos caracteres especiales (se han cogido algunos de los muchos disponibles), que son los que se indican en la parte inferior de la Figura 1.

   El supuesto que se ha considerado para configurar las expresiones regulares es que el formato válido de entrada para identificar el correo y su contraseña es que aparezca el correo en el formato adecuado, después el separador mencionado y justo después la contraseña.

2 Expresiones regulares usadas en el programa

Como ya se ha referido, se han usado tres expresiones regulares para identificar el correo, el separador (en cualquiera de sus variantes, evidentemente) y la contraseña, en ese orden.

La explicación de lo que significan las expresiones regulares se muestra en la figura 1:

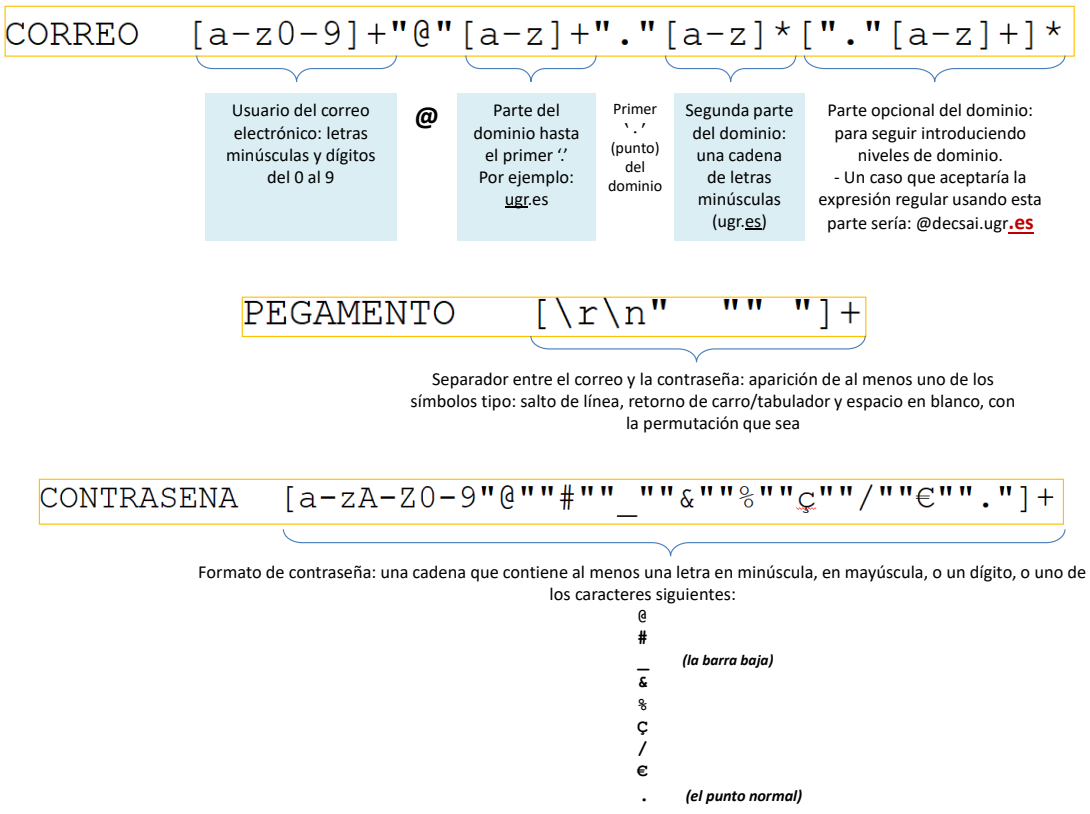

Figura 1: Descripción de las expresiones regulares

El formato de cada expresión regular, para el correo, el separador (que le hemos llamado simbólicamente PEGAMENTO) y la contraseña son los indicados en la anterior figura 1.

3 Procesamiento de las cadenas

De esta forma, el patrón que identifica al correo y la contraseña en el fichero de entrada tiene la siguiente forma, según el fichero fuente:

```
{CORREO}{PEGAMENTO}{CONTRASENA} {
   string cadena(yytext);
   string contrasena;

   cout << cadena << "\n";
   size_t pos = cadena.length(), aux;

   aux = cadena.find("\n");
   if (aux != string::npos && aux < pos){
      pos = aux;
   }

   aux = cadena.find("\r");
   if (aux != string::npos && aux < pos)
      pos = aux;

   aux = cadena.find(" ");          //TAB
   if (aux != string::npos && aux < pos)
      pos = aux;

   aux = cadena.find(" ");          //ESPACIO
   if (aux != string::npos && aux < pos)
      pos = aux;

   // anadir correo a la lista
   correos.push_back(cadena.substr(0, pos));

   // empezamos a procesar la contrasena
   contrasena = cadena.substr(pos+1);

   pos = contrasena.find_last_of("\n");

   if (pos != string::npos)
      contrasena = contrasena.substr(pos+1);

   pos = contrasena.find_last_of("\r");
   if (pos != string::npos)
      contrasena = contrasena.substr(pos+1);

   pos = contrasena.find_last_of(" ");          //TAB
   if (pos != string::npos)
      contrasena = contrasena.substr(pos+1);

   pos = contrasena.find_last_of(" ");    //ESPACIO
   if (pos != string::npos)
      contrasena = contrasena.substr(pos+1);

   // anadir contrasena
   contrasenas.push_back(contrasena);
}
```

En este fragmento de código lo que hacemos es, partiendo de la cadena que se ajusta a la expresión regular {CORREO}{PEGAMENTO}{CONTRASENA}, identificada dentro del código como yytext, tal y como se indicaba en el guión, separar, por un lado, el correo y por el otro, la contraseña.

El hecho de aceptar la concatenación de múltiples separadores en lugar de uno sólo dificulta ligeramente el código, por lo que explicamos su funcionamiento a continuación. Teniendo en cuenta que el correo viene antes del separador PEGAMENTO, lo que hemos hecho aquí es buscar la posición del primer carácter de todos los que pueden actuar como separador, que son los que aparecen en la expresión regular de PEGAMENTO: salto de línea, retorno de carro, espacio y tabulador.

Una vez se ha guardado el correo cogiendo la parte de la cadena que va desde el inicio hasta la posición de dicho primer carácter separador, hay que realizar el proceso contrario con contraseña. Es decir, partiendo de lo que resta de quitarle a la cadena original el correo, se busca la posición del último carácter separador, pues la contraseña va a la derecha de PEGAMENTO. En realidad, en este código se realiza un proceso ligeramente distinto pero equivalente, que consiste en ir desplazando la cadena `contrasena` hasta que esta empiece un carácter más allá del último carácter de la cadena `PEGAMENTO`.

Una vez hecho esto, ya tenemos el correo y su contraseña asociada ocupando las mismas posiciones en sus respectivos vectores correos y contraseñas, de manera que quedan relacionados por la posición que ocupan.

3.1 Ordenación de los correos y contraseñas

Una vez que se ha terminado de leer el fichero de entrada, se tienen completamente rellenados dos vectores, uno con los correos (correctamente almacenados) y otro con las contraseñas. El siguiente paso será ordenar cada par $\{correo, contrasena\}$ por el orden alfabético del dominio, como ya se ha referido. Para ello se usa el algoritmo de la burbuja, que se muestra a continuación, ya incluido en nuestro programa:

```
1  void ordenarCorreos(){
2        vector<string> dominios;
3        int pos;
4        string tmp;
5        bool intercambio = true;
6
7        for (int i=0; i<correos.size(); i++){
8              pos = correos[i].find("@");
9              dominios.push_back(correos[i].substr(pos+1));
10       }
11
12       // ordenacion de los correos, contrasenas y dominios usando el criterio
13       // de ordenacion de los dominios alfabeticamente
14       // (se ha usado el método de la burbuja)
15       for (int i=0;i<dominios.size() && intercambio;i++){
16             intercambio=false;
17             for(int j=dominios.size()-1;j>i;j--)
18                   if(dominios[j]<dominios[j-1])
19                   {
20                         intercambio=true;
21                         tmp=dominios[j];
22                         dominios[j]=dominios[j-1];
23                         dominios[j-1]=tmp;
24                         tmp=correos[j];
25                         correos[j]=correos[j-1];
26                         correos[j-1]=tmp;
27                         tmp=contrasenas[j];
28                         contrasenas[j]=contrasenas[j-1];
29                         contrasenas[j-1]=tmp;
30                   }
31       }
32 }
```

En esta función se implementa la siguiente funcionalidad, por orden:

- Se crea y rellena un vector con los dominios de los correos encontrados (`ugr.es`, `yahoo.es`, etc.)
- A continuación se procede a ordenar ese vector alfabéticamente, y a la vez se van ordenando los vectores de correos y contraseñas, aprovechando el algoritmo.

De esta forma, cuando acabe el procedimiento, tendremos los dos vectores globales que nos interesan (correos y contraseñas) ordenador por los dominios de los primeros.

3.2 El fichero de salida

El último paso es montar el fichero de salida. Este fichero en nuestro caso tiene un esquema muy sencillo:

> El contenido del fichero se organiza en: un correo por cada línea, y en la línea siguiente a cada correo se muestra su contraseña correspondiente. Evidentemente, los correos y sus respectivas

contraseñas están ordenados según lo descrito anteriormente, de principio al fin del fichero de salida.

La función que se encarga de formatear el fichero de salida es:

```
// flujos de entrada y salida (declarados en la parte global)
ifstream fichero;
ofstream out;

void mostrarResultados(){
      ordenarCorreos(); // ordenas vectores de correos y claves

      // rellenar el fichero de salida con tuplas <correo,clave> ordenadas
      //  por nombre de dominio según el orden alfabético. Cada correo/clave
      // en una línea del fichero, y cada clave tiene justo encima su correo
      for (int i=0; i<correos.size(); i++){
            out << correos[i] << "\n";
            out << contrasenas[i] << "\n";
      }
}
```

Un ejemplo de fichero de salida, dado el fichero de entrada que hemos proporcionado en la entrega de PRADO, sería (un fragmento):

```
quetepareceelprograma@correo.ugr.es
mepareceinteresante
miracomofunciona@decsai.ugr.es
funcionamientocorrecto
loimportanteesquefuncionecorrectamente@decsai.ugr.es
funcionara
correoprueba@gmail.com
todocorrecto
elprogramafuncionara@gmail.com
losabia
datuopinion@gmail.com
yanolades
esteprogramamola@hotmail.com
enrealidadfallara
proando123@hotmail.com
probando123
correo3@hotmail.com
otraprueba
correo2@yahoo.es
nuevocorreo
poahorafunciona@yahoo.es
noshacostadounpoco
```