Creación tarjetas de empresa con `flex`

Carmen Azorín Martí (48768328W)
Pedro Haimar Castillo García (77243253F)

Índice

1. Presentación del problema

En este proyecto hemos realizado un programa que realiza tarjetas de empresa. La idea es tomar un documento en el que aparezcan los detalles de diferentes empresas, cinco datos ordenados con sus respectivas restricciones:

1. **Nombre de empresa**: Palabra o conjunto de palabras. Puede contener números intermedios.

   Ejemplo. G2 Esports

2. **Descripción**: Palabra o pequeño conjunto de palabras que explican qué ofrece la empresa.

   Ejemplo. Equipo Gaming

3. **Número de teléfono**: El número de teléfono de cualquier empresa de España debe tener 9 dígitos. Además, el primero de ellos debe estar entre 6 y 9.

   Ejemplo. 812 088 383

4. **Correo electrónico**: El correo electrónico de una empresa debe contener como máximo un carácter '@'.

Ejemplo. info@g2esports.com

5. **Página web**: La página web es una cadena texto con subdominio o protocolo opcionales seguido de '.' y seguido del dominio.

   Ejemplo. https://g2esports.com/

Para cada una de las empresas que tenga los datos correctamente introducidos, se creará un archivo que contendrá la tarjeta para dicha empresa.

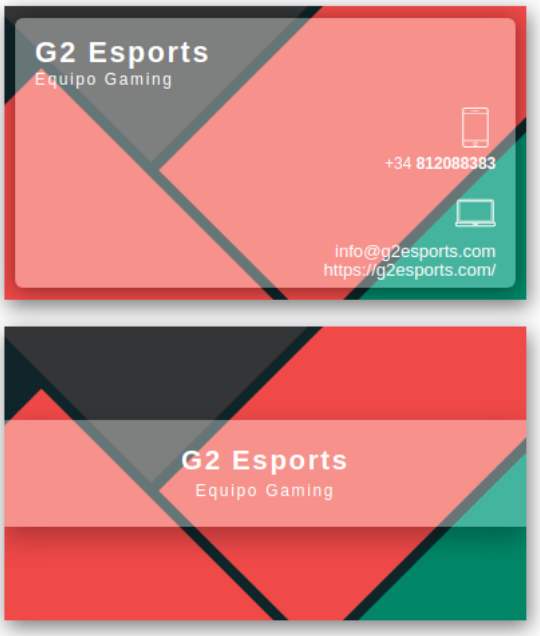

Figura 1: Tarjeta para G2 Esports

2. Solución

Para realizar este programa hemos utilizado dos recursos: `flex++` y `html`.

En primer lugar, hemos creado un programa escrito en `flex` que leerá el documento donde aparecen los datos de una empresa en el orden especificado anteriormente y separados por uno o más tabuladores. Cada empresa corresponderá con una línea del documento.

Se buscará la línea formada por una expresión regular que coincida con lo fijado, es decir: un nombre de empresa, tabulador(es), descripción, tabulador(es), número de teléfono, tabulador(es), correo electrónico, tabulador(es) y página web. Una vez se encuentre, en la plantilla de tarjeta escrita en `css` generada en el main, se incluirán los datos con el documento `html` asociado.

Por otro lado, si la línea no coincide con la expresión regular buscada, intentará averiguar cuál es el dato o los datos que no corresponden con los requisitos. Por ejemplo, si la

empresa de la línea 4 del documento no tiene el número de teléfono correcto, se indicará por pantalla.

Cabe destacar que los programas de `html` y `css` son de Internet, de una página dedicada a realizar plantillas para tarjetas de empresa. Nosotros nos hemos basado en ese código y, haciendo pequeños cambios, hemos conseguido nuestro objetivo. Esta página web de la que hablamos es `https://codepen.io/SRHubli/pen/djdPqz`.

3. Sección de declaraciones

En esta sección hay dos partes: cabecera del archivo `c++` y las expresiones regulares creadas.

En la cabecera del archivo nos encontramos con las declaraciones de 9 variables y 2 funciones.

La variable fichero será la que guarde que el documento del que vamos a leer las empresas línea por línea. Las siguientes 5 variables son del tipo *string*, por cada empresa que contenga todos los datos correctamente indicados, se incluirán sus datos en dichas variables. Las variables error y revisa, se utilizarán para imprimir por pantalla que ha habido algún error con respecto a las expresiones regulares. Para terminar, la variable contador llevará la cuenta de qué número de línea (empresa) se está leyendo, para indicar aquellas empresas que contengan algún dato erróneo.

La función GenerarArchivo sirve para crear un documento `html` en el que se guardará la tarjeta de la empresa que llame a esta función. Recibe como parámetros los 5 datos que puede tener una empresa. Se llamará a esta función si, y sólo si, toda la información de la empresa cumple con los requisitos.

La función GenerarCarta sirve para crear un documento `css` que definirá el estilo de la tarjeta. A esta función también se llamará cada vez que una empresa tenga todos sus datos correctos.

En cuanto a las expresiones regulares, hemos creado una para cada dato:

1. NOMBRE_EMPRESA: `[A-Za-z]([A-Za-z0-9]|‘‘ ’’)*`

   `[A-Za-z]` sirve para indicar que el primer carácter debe ser una letra, mayúscula o minúscula.

   `([A-Za-z0-9]|‘‘ ’’)*` sirve para indicar que el nombre de la empresa puede contener letras y números, incluso puede estar compuesta de varias palabras gracias al espacio.

2. DESCRIPCION: `[A-Z]([a-z]|‘‘ ’’[A-Za-z])*`

`[A-Z]` sirve para indicar que el primer carácter debe ser una letra mayúscula.

`([a-z]|‘‘ ’’[A-Za-z])*` sirve para indicar que la descripción debe estar formada sólo por letras. Las letras mayúsculas solo se escribirán a principio de palabra.

3. NUMERO_TELEFONO: `[6-9]([0-9]{8})`

   `[6-9]` indica que el primer dígito de un número de teléfono español debe empezar por 6, 7, 8 ó 9.

   `([0-9]{8})` indica que debe haber 8 dígitos después del primero, es decir, 9 dígitos en total forman un número de teléfono español.

4. CORREO_ELECTRONICO: `[a-z]([a-z0-9]| \.)*@[a-z]([a-z0-9]|\.)* \.[a-z]+`

   `[a-z]` indica que el correo electrónico debe empezar por letra minúscula.

   `([a-z0-9]| \.)*` indica que el nombre de usuario está formado por letras minúsculas, números y puntos.

   `@[a-z]([a-z0-9]| \.)*` indica que el dominio comienza por un @ y va seguido de letras minúsculas, números y puntos.

   `\.[a-z]+` indica que el dominio debe terminar con un punto seguido de letras minúsculas.

5. PAGINA_WEB: `(www\.|http[s]?:\/\/)[a-z]([\.]?[a-z0-9])*\.[a-z]+[\/]?`

   `(www\.|http[s]?:\/\/)` indica que el subdominio debe ser www., http:// o https://

   `[a-z]([\.]?[a-z0-9])*` indica que el dominio comienza por letra minúscula y va seguido de más letras minúsculas, números o puntos. Cada punto debe ir seguido de una letra minúscula o un número.

   `\.[a-z]+[\/]?` indica que el dominio acaba con un punto seguido de letras minúsculas y una barra al final opcional.

Además, hemos creado la expresión regular TABULADOR que acepta uno o mś tabuladores

Una vez hecho esto, sabemos que la línea de una empresa será correcta si cumple que

```
{NOMBRE_EMPRESA}{TABULADOR}{DESCRIPCION}{TABULADOR}{NUMERO_TELEFONO}
                                {TABULADOR}{CORREO_ELECTRONICO}{TABULADOR}
                                {PAGINA_WEB}
```

es decir, si cumple el orden de los datos y los requisitos de cada dato. A esta expresión la hemos denominado LINEA.

Aparte, para conseguir averiguar qué dato o datos están mal en una línea, hemos creado una expresión regular para cada fallo que se podría dar. Por ejemplo, si solamente falla el nombre de la empresa, la línea cumplirá la siguiente expresión regular

```
{CUALQUIER_PALABRA}{TABULADOR}{DESCRIPCION}{TABULADOR}{NUMERO_TELEFONO}
                                {TABULADOR}{CORREO_ELECTRONICO}{TABULADOR}
                                {PAGINA_WEB}
```

Otro ejemplo podría ser si en una línea falla el nombre de la empresa, la descripción y el número de teléfono. Entonces, dicha línea cumplirá la siguiente expresión

```
{CUALQUIER_PALABRA}{TABULADOR}{CUALQUIER_PALABRA}{TABULADOR}{CUALQUIER_PALABRA}
                                {TABULADOR}{CORREO_ELECTRONICO}{TABULADOR}
                                {PAGINA_WEB}
```

4. Sección de reglas

En cuanto a las reglas, hemos creado dos que sólo sirven para que no salgan errores falsos: cuando se lea un tabulador o un salto de línea, que no ocurra nada.

El *string* que cumpla la expresión de LINEA será una empresa con todos los datos correctos. Este *string* se guardará en el vector `yytext[]` de caracteres. Sabemos que los datos estarán separados por tabuladores, así que el primer dato antes de un tabulador, será el nombre de la empresa, que lo guardaremos en la variable nom_emp declarada anteriormente. Tras los tabuladores, nos encontraremos con la descripción de la empresa, que lo guardaremos en la variable desc. Volvemos a ignorar los tabuladores y guardamos el número de teléfono en la variable num_telf. Ya para terminar, entre tabuladores de nuevo, encontramos el correo electrónico de la empresa que se guardará en la variable correo y la página web de la empresa, que se guardará en pag_web.

Una vez tengamos todos los datos guardados en sus respectivas variables, podemos llamar a GenerarArchivo y pasarle éstos datos como parámetros. Ya tendremos la tarjeta de ésta empresa creada en un documento de la carpeta donde se esté ejecutando el programa.

Finalmente, volvemos a inicializar las variables a *string* vacíos para que se lean los datos de la siguiente empresa que `flex` encuentre. Además, añadimos uno al contador, porque se ha encontrado una nueva empresa.

Por otro lado, tenemos las expresiones de fallo de alguno de los datos. Cada vez que una línea corresponda con una de estas expresiones, se imprimirá por pantalla qué número de empresa es la que está dando este problema y qué datos son los erróneos.

5. Sección de procedimientos

Hablemos ahora de los procedimientos que podemos encontrar dentro de nuestro programa `flex`. Tal y como se ha explicado anteriormente, hay dos procedimientos aparte del main.

Por un lado, GenerarCarta() es un procedimiento que lo que hace es generar un archivo `style.css` en el directorio de trabajo. La manera de hacer este "paso de lenguajes de programación", por llamarlo de alguna forma, es simple. Se basa en ir pegando a un string código tal cual de nuestro fichero (en nuestro lenguaje que queramos) y cada vez que encontremos una variable de nuestro fichero que queremos que tenga un valor en específico de nuestro programa, pegamos al string la variable de nuestro programa y luego seguimos pegando a nuestro ya redundante string el código a partir de ahí hasta encontrar otra vez una variable del fichero que queremos que coincida con el programa o hasta que llegue al fin.

De manera similar se puede explicar el funcionamiento de la función GenerarArchivo(), admitiendo esta 5 parámetros y por lo tanto generando archivos html en los que hay 5 variables que han sido pasadas de nuestro programa al archivo html. Una idea bastante útil que probablemente nos servirá en el futuro es la de generar archivos de un lenguaje desde otro lenguaje y con un plus de variables compartidas.

Hablando ahora del main, vemos que nada más empezar comprobamos si hay dos argumentos para salir en caso contrario. Acto seguido verificamos si hemos podido abrir correctamente el archivo. Primero llamamos a GenerarCarta() que ya hemos visto previamente que hace, luego se crea el flujo `flex` que vamos a usar para leer el fichero y finalmente con el metodo .yylex() empezamos a buscar expresiones regulares de la forma declarada previamente en el programa en nuestro fichero.

6. Compilación y ejecución

En cuanto a compilación y ejecución no tiene mucho misterio: El comando
`flex++ archivo`
nos genera un fuente en `C++` a partir de un fuente en formato lex. Luego usamos
`g++ lex.yy.cc -o OutputName`
para compilar nuestro fuente en `C++` y generar así nuestro programa finalmente. En cuanto

a la ejecución, bastará con ejecutarlo de la forma
`./OutputName fichero`
donde fichero guarda en nuestro caso todas las empresas que queremos probar a ver qué campos le fallan.

Referencias

1. Manual de `flex`

   $http://webdiis.unizar.es/asignaturas/LGA/docs_externos/flex-es-2{,}5.pdf$

2. Plantilla tarjetas

   $https://codepen.io/SRHubli/pen/djdPqz$