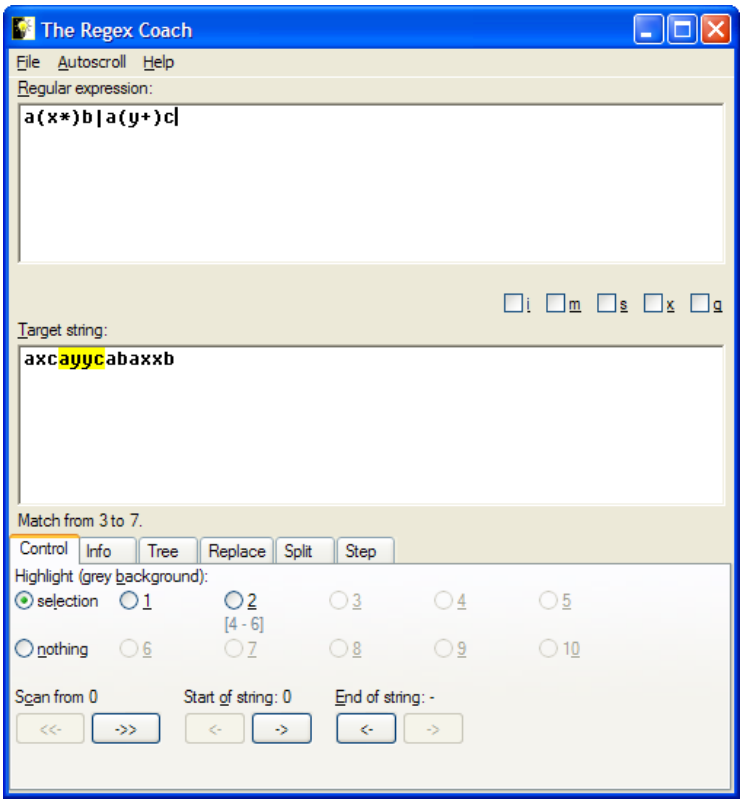
PRÁCTICA 2: Expresiones Regulares
Semana: 22 al 26 de septiembre de 2014

1. Objetivo

En esta práctica comprobaremos uno de los usos de las expresiones regulares: el análisis léxico. Examinaremos un fichero que contiene un programa escrito en Java e identificaremos sus distintos componentes léxicos (identificadores, números, palabras reservadas,...). Para ello utilizaremos la herramienta Regex Coach. Lo primero que debe hacer es ejecutar la aplicación. A continuación, cargue el fichero sort.java (puede bajarlo de la página web de la asignatura y guardarlo en un directorio local). Con este fichero trabajaremos a lo largo de esta práctica escribiendo expresiones regulares para buscar cadenas en él.

2. Herramienta Regex Coach

La ventana que se muestra tiene dos paneles: el panel de expresiones regulares (arriba) y el panel de introducción de cadena (abajo). Para buscar en el texto, en primer lugar sitúe el cursor en el panel superior y escriba la expresión regular a buscar. Sitúese en el panel inferior y escriba la cadena sobre la que buscar. Verá que, a medida que escribe, la aplicación colorea con amarillo la subcadena que encaja con la expresión regular escrita. También puede buscar la correspondencia de la expresión regular en la cadena (o parte de la cadena) escrita pulsando los botones de nombre Scan.



3. Expresiones regulares

3.1. Expresiones regulares básicas

Las expresiones más sencillas son las cadenas de caracteres. Si escribe una cadena como if aparecerá resaltada la primera aparición de la cadena en el texto. Pulsando el botón Scan avanzará a las siguientes apariciones. Fíjese que, por defecto, la búsqueda considera que las letras en mayúsculas concuerdan también con las letras en minúsculas de la búsqueda. Por ejemplo, si en el texto aparece "iF", "If" o "IF" también aparecerán resaltadas cuando busque "if".

También debe darse cuenta de que todo el texto está considerado como una sola cadena y lo que se hace es resaltar aquellas subcadenas que responden a la expresión de búsqueda. Así, no es necesario que "if" aparezca como una cadena completa, sino que si en el programa hay una variable que se llama "periferia", las letras "i" y "f" también aparecerán resaltadas.

Ejercicio 1

Comprobar el funcionamiento de la búsqueda de cadenas encontrando todas las apariciones de las palabras reservadas "else", "public" y "void", ...

- ¿Cuántas veces aparece cada palabra reservada?
- ¿Qué expresión regular ha utilizado para la búsqueda?
- ¿Qué sucede si la cadena que introduce no está en ningún lugar del texto?

3.2. Disyunción de expresiones regulares

Una de las operaciones que podemos realizar con expresiones regulares es la unión. En Regex la unión o disyunción se expresa mediante el símbolo **|**. Así, una expresión regular que denota a las cadenas "if" y "else" sería**: if | else**

Este operador se puede utilizar varias veces, como en **public | private | protected** que denota las 3 cadenas "public", "private" y "protected". Comprobar su funcionamiento.

Ejercicio 2

Escribir una expresión regular que denote los tipos de datos simples en Java (**int, float, char, double, …)** y comprobar su funcionamiento.

3.3. Concatenación de expresiones regulares

Para expresar la concatenación de expresiones regulares basta con colocar dos expresiones regulares una a continuación de la otra. En el caso de que las expresiones regulares contengan operadores (como por ejemplo el de disyunción) es posible que se necesiten paréntesis para poder establecer las prioridades de evaluación. Como ejemplo de uso de paréntesis y concatenación podemos ver la expresión:

(sulf|clor)(ito|ato)

que denota las cuatro cadenas "sulfito", "sulfato", "clorito" y "clorato".

3.4. Operadores de repetición

- El operador de repetición ***** o *clausura de Kleene* se aplica a la expresión regular inmediatamente anterior. Ej: la expresión: **ba*** denota las cadenas que comienzan por "b" y luego tienen una "a" o ninguna o muchas.

- El operador "**.**" denota cualquier carácter. Ej: si tenemos un fichero con nombres de archivo y queremos encontrar aquellos que terminan en "txt" podríamos escribir: **.*txt**

- El operador **+** es parecido al operador * pero requiere que la expresión precedente se repita al menos una vez. Ej: para la expresión **ba+** entonces "ba", "baa", "baaa" … son cadenas válidas, pero no "b".

- El operador "**?**" sirve para indicar que una cadena es opcional, es decir, que puede aparecer una vez o ninguna. Ej: **cama(feo|leon)?** denota las palabras "camafeo" y "camaleon" y también la palabra "cama"

- Para indicar exactamente el número de repeticiones de una cadena se consigue mediante el operador **{n}** o **{n,m}** donde n indica el mínimo número de repeticiones y m el máximo. Por ejemplo:
  - x{4} denota las cadenas que tienen exactamente 4 'x'.
  - x{4, } denota las cadenas que tienen al menos 4 'x'.
  - x{4,5} denota las cadenas que tienen al menos 4 'x' y como mucho 5.

- Para definir conjuntos de caracteres se utilizan los corchetes "**[**" y "**]**". Por ejemplo:
  - Mediante [1357] representamos cualquier carácter 1, 3, 5, 7.
  - Con [a-z] representamos el rango de caracteres entre la "a" y la "z" en ASCII.

- Si lo que queremos es denotar todos los caracteres fuera de un rango usaremos el operador ^. La expresión [^0-9] denota cualquier carácter que no sea un dígito.

Ejercicio 3: Escribir expresiones regulares para los siguientes casos:

3.1 Buscar los identificadores de variables en Java (cadenas formadas por caracteres alfabéticos, dígitos, $ o subrayado y que no empiecen por un número.

3.2 Buscar una expresión regular que denote cualquier cadena compuesta por alguno de los siguientes caracteres: x, y, z.

3.3 Buscar una expresión regular que denote los números enteros mayores que 99.

Ejercicio 4:

En las tablas que siguen, cada fila de la tabla corresponde a una expresión regular; cada columna, a una cadena de caracteres. Completar las tablas escribiendo SÍ o NO en cada una de sus celdas, según si la cadena que encabeza la columna pertenece o no al lenguaje representado por la expresión regular que encabeza la fila.

	λ	**bca**	**xxx**	**x**	**(xxx)**	**(z**
[^abc]?						
([^a][^b][^c])*						
[^(abc)]+						
\([^abc]\)						

	()	**(01)**	**(^ab)**	**01**	**\(01\)**	**.)**	**(01)***	**abab)**	***)**
\((^ab)*\)									
(01)*									
[^.].									
[(ab)*]\)									