**Practica No. 2 Expresiones regulares en lenguajes de programación**

**20/02/2020**

**Integrantes:**Diego Jasso Carrillo  
Luis Heriberto Renovato Espinosa  
Adalberto Gabriel Delgado Trujillo

1. **Herramientas para manejar expresiones regulares**

**Esta parte fue realizada por:** Diego Jasso Carrillo**.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Herramienta** | **Lenguaje de programación** | **Características** | **Forma de uso** | **Elementos importantes** |
| Regex  (java.util.regex) | Java | - Proporciona una serie de clases para poder hacer uso de la potencia de este tipo de expresiones en Java.  - El paquete  java.util.regex está formado por dos clases, la clase Matcher y la clase Pattern y por una excepción,  PatternSyntaxException.  - La clase Pattern es la representación compilada de una expresión regular, mientras que la clase Matcher crea un objeto con el cual puedes hacer operaciones sobre la secuencia de caracteres que queremos validar. | - Creamos patrones que deben ser compilados, a partir de estos creamos objetos Matcher para poder realizar las operaciones sobre la cadena en cuestión. Ejemplo:  Pattern patrón = Pattern.compile("camión");  El método Pattern devuelve la expresión regular que hemos compilado, el método Matcher crea un objeto Matcher a partir del patrón  - El método Pattern devuelve la expresión regular que hemos compilado, el método Matcher crea un objeto Matcher a partir del patrón. | La herramienta cuenta con los siguientes métodos con los cuales se puede verificar la cadena que queramos:  **matches**: Compara el patrón con la secuencia sugerida (en este caso compara el patrón “camión” con la secuencia “camión” si encajaría, y si usamos la secuencia “mi camión es verde” no encajaría con el patrón.  **lookinAt**: intenta encajar la secuencia con el patrón (si la secuencia es “mi camión es verde” encajaría perfectamente con el patrón).  **find**: va buscando subcadenas dentro de la cadena de caracteres que cumplan con el patrón compilado.  Todos estos métodos regresan un booleano que indican que la operación ha tenido éxito. |
| Glibmm  #include <glibmm/regex.h> | C++ | Las funciones **[Glib :: Regex](https://developer.gnome.org/glibmm/stable/classGlib_1_1Regex.html" \o "Expresiones regulares compatibles con Perl: combina cadenas con expresiones regulares. )** implementan la coincidencia de patrones de expresión regular utilizando sintaxis y semántica similar a la expresión regular | Incluya el encabezado glibmm, más giomm si es necesario:  #include <glibmm.h>  #include <giomm.h>  (Puede incluir encabezados individuales, como en su glibmm/ustring.hlugar).  Si su archivo fuente es program.cc, puede compilarlo con:  g ++ program.cc -o program `pkg-config --cflags --libs glibmm-2.4 giomm-2.4`  Alternativamente, si usa autoconf, use lo siguiente en configure.ac:  PKG\_CHECK\_MODULES ([GLIBMM], [glibmm-2.4 giomm-2.4])  Luego use las variables generadas GLIBMM\_CFLAGSy GLIBMM\_LIBSen los archivos del proyecto Makefile.am.  Por ejemplo:  program\_CPPFLAGS = $ (GLIBMM\_CFLAGS)  program\_LDADD = $ (GLIBMM\_LIBS) | * [**Glib :: ustring**](https://developer.gnome.org/glibmm/stable/classGlib_1_1ustring.html) : una clase de cadena UTF-8 que se puede usar de manera intercambiable con [std :: string](http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a01512.html" \l "ga32db3d9898c44d3b3a578b560f7758cc) .   Más [**Funciones de utilidad de cadena**](https://developer.gnome.org/glibmm/stable/group__StringUtils.html).   * [**Glib :: RefPtr**](https://developer.gnome.org/glibmm/stable/classGlib_1_1RefPtr.html) : un puntero inteligente de conteo de referencias, para usar con **[Glib :: ObjectBase](https://developer.gnome.org/glibmm/stable/classGlib_1_1ObjectBase.html" \o "Glib :: ObjectBase es una clase base común para objetos e interfaces. )** o similar * [**Conversión de juego de caracteres**](https://developer.gnome.org/glibmm/stable/group__CharsetConv.html) * [**Glib :: Regex**](https://developer.gnome.org/glibmm/stable/classGlib_1_1Regex.html) : coincidencia de cadenas de expresión regular. * [**Glib :: KeyFile**](https://developer.gnome.org/glibmm/stable/classGlib_1_1KeyFile.html) : análisis y escritura de archivos clave (similar a los archivos .ini) * [**Glib :: Checksum**](https://developer.gnome.org/glibmm/stable/classGlib_1_1Checksum.html) * [**Glib :: Fecha**](https://developer.gnome.org/glibmm/stable/classGlib_1_1Date.html) , **[Glib :: Temporizador](https://developer.gnome.org/glibmm/stable/classGlib_1_1Timer.html" \o "Interfaz de cronómetro portátil. )** , **[Glib :: TimeVal](https://developer.gnome.org/glibmm/stable/structGlib_1_1TimeVal.html" \o "Glib :: TimeVal es un contenedor alrededor de la estructura glib GTimeVal. )** * [**Glib :: Dispatcher**](https://developer.gnome.org/glibmm/stable/classGlib_1_1Dispatcher.html) : comunicación entre hilos |

**Esta parte fue realizada por:** Adalberto Gabriel Delgado Trujillo

|  |  |  |
| --- | --- | --- |
| Componente Léxico | Descripción informal | Expresión regular |
| ***For*** | F seguido de O seguido de R | FOR |
| ***Break*** | B seguido de R seguido de E seguido de A seguido de K | BREAK |
| ***>=*** | Operador racional, > seguido de = | >= |
| ***{}*** | { seguida de } | {} |
| ***;*** | Separador de sentencias | ; |
| ***AnTeNa*** | Identificador, carácter alfabético pero no un digito, A seguida de n seguida de T seguida de e seguida de N seguida de a | AnTeNa |
| ***True*** | Indica valores verdaderos,  T seguida de r seguida de u seguida de e | True |
| ***Mensaje*** | Cadena vacía de caracteres, M seguido de e seguido de n seguido de s seguido de a seguido de e | mensaje |

**Esta parte fue realizada por**: Luis Heriberto Renovato Espinosa

|  |  |
| --- | --- |
| **for** | |
| Cadena valida; for. |  |
| Cadena invalida; f |  |
| Cadena invalida; fo |  |
| Cadena invalida; rof |  |

|  |  |
| --- | --- |
| **break** | |
| Cadena valida; break |  |
| Cadena invalida; bre |  |
| Cadena invalida;  bek |  |

|  |  |
| --- | --- |
| **Mensaje** | |
| Cadena valida; //hola |  |
| Cadena invalida; // HOLA |  |
| Cadena Invalida; //123 |  |

|  |  |
| --- | --- |
| **AnTeNa** | |
| Cadena valida; AnTeNA |  |
| Cadena invalida; 12345 |  |
| Cadena invalida; antena134 |  |

|  |  |
| --- | --- |
| **True** | |
| Cadena valida; true |  |
| Cadena invalida; tu |  |
| Cadena invalida ; re |  |

**Conclusiones:**

* **Diego**: Lo que aprendí de esto es que en java es más fácil de trabajar o manejar las expresiones regulares, yo no sabía que existía una librería para trabajar con dichos elementos, además de que es muy fácil de hacerlo y tiene los métodos necesarios para identificarlos. Mientras que en c++ pude ver que es un poco más complicado (ya que no conozco muy bien el lenguaje) además fue muy difícil encontrar otra herramienta, ya que donde buscara me recomendaba regex que es de java. Para los diferentes componentes léxicos que existen en java pude observar que algunos tienen una descripción informal y expresión regulas bastante simple, pensé que sería un poco más complejas pero no.
* **Luis**: En este trabajo pude por primera vez manejar las expresiones regulares en java, ya que solo las pude utilizar en HTML, ahí me di cuenta que casi se declaran igual solo que en java hay una librería que te ayuda en eso de las expresiones que es java.util.regex. En HTML la declarabas en las etiquetas y eso hizo que me confundiera mucho pero ya después de ver un video en YouTube de cómo hacer expresiones regulares en java todo me quedo un poco más claro.
* **Adalberto**: En esta práctica podemos diferenciar la utilidad de los diferentes tipos de argumentos léxicos, así como saber su expresión regular y el para qué sirven dando una explicación informal, a si podemos diferenciarlos y comprobar a qué cadena pertenece cada carácter para formar un alfabeto.