Análisis Léxico.
Sesión 1: Generación de Scanners utilizando JLex

Autómatas y Matemáticas Discretas
Escuela de Ingeniería Informática
Universidad de Oviedo
curso 2016-2017

1. Análisis léxico

El análisis léxico tiene por objetivo procesar unas cadenas de caracteres, escritas en un determinado lenguaje, y separarlas en palabras relevantes. Las cadenas de caracteres pueden introducirse directamente por el teclado o bien estar almacenadas en un archivo. Por ejemplo, si el archivo de entrada al analizador léxico (scanner) contiene el siguiente texto:

```
diferencia = (beneficio-coste)
```

Entonces el scanner leerá la siguiente secuencia de caracteres:

```
\tdiferencia = (beneficio-coste)
```

El objetivo del análisis léxico es extraer las palabras significativas del texto de entrada (denominadas **Token**, en plural *Tokens*), eliminando las que carecen de significado -espacios en blanco, tabuladores ($\backslash t$), etc-.

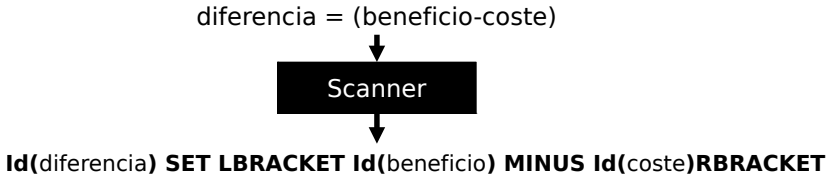
1.1. ¿A qué se denomina *Token*?

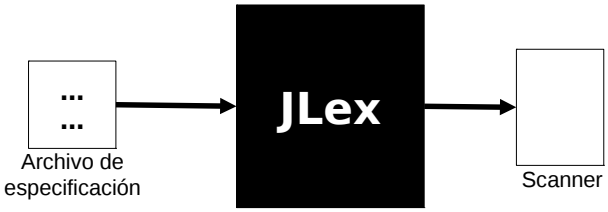
Un **token** es una categoría sintáctica del lenguaje. Por ejemplo, en castellano tenemos los sustantivos, verbos, adverbios, etc. Cada lenguaje de programación tiene sus propias categorías sintácticas, y son definidas en la *especificación del lenguaje*; por ejemplo: *Identificador*, *Número*, *Keyword*, etc. A lo largo de este curso se utilizará la especificación de un lenguaje dado por el equipo de profesores.

1.2. ¿Qué es un *Analizador Léxico* o *Scanner*?

Un **analizador léxico** o **Scanner** es un programa que transforma la cadena de caracteres que recibe como entrada -bien sea desde teclado o desde un archivo- en la correspondiente secuencia ordenada de tokens.

Cada token consiste al menos en un **identificador** de categoría sintáctica -keyword, operador, identificador, constante según el tipo de dato, símbolos de puntuación, etc.- y, dependiendo del tipo de identificador, **información** extra acerca del patrón encontrado, como se puede observar en la siguiente figura.





2. Ejercicios

Conéctate mediante un cliente *ssh* (p.e., putty) a la máquina *ritchie* (con ip 156.35.94.1). Según te conectes con tu UO y contraseña, estarás en tu directorio raíz.

2.1. Obtener los archivos básicos

1. Si aún no tienes creado el directorio `practicasAMD` puedes crearlo mediante la oreden `mkdir practicasAMD`. Cámbiate al nuevo directorio y y crea el directorio `LexicalAnalysis`. De nuevo cámbiate al nuevo directorio y y crea el directorio `Training`. Muévete a tu directorio `~/practicasAMD/LexicalAnalysis/Training`. Si ejecutas el comando **pwd** comprobarás tu directorio de trabajo actual es:

   ```
   /home/students/yourUO/practicasAMD/LexicalAnalysis/Training
   ```

2. Copia al directorio actual todos los archivos y directorios disponibles en:
   /var/asignaturas/AMD/practicas/2016-2017/LexicalAnalysis/Training/
   mediante el siguiente comando (no olvides el punto al final):

   ```
   cp -r /var/asignaturas/AMD/practicas/2016-2017/LexicalAnalysis/Training/* .
   ```

3. Muévete al directorio **exercise1** y comprueba con el comando **ls** que se dispone de los archivos *README* (con información de los archivos del directorio), *Scanner.class* (el programa en java que realiza la tarea de analizador léxico), *test* (archivo de prueba, conteniendo algunas cadenas de caracteres), *Yylex.class* y *Yytoken.class* (archivos de java que usa nuestro scanner).

2.2. Análisis del archivo de texto con el scanner

Nuestro objetivo es aprender a utilizar el *scanner* (programa escrito en java) con un texto de entrada almacenado en un archivo, e identificar cada mensaje de salida con el correspondiente texto de entrada.

El programa `Scanner.class` reconoce los siguientes `tokens`:

- Las palabras reservadas **begin**, **end** y **print**
- Números enteros definidos como una secuencia de al menos un dígito.
- Paréntesis y el punto y coma (;).
- Operadores: $+, -, *, /, <, >, ==$.
- Espacios en blanco, tabuladores y el carácter de nueva línea. Estos tokens a diferencia de los anteriores, para los que el analizador muestra un mensaje, son ignorados y no se muestra ningún mensaje.
- Para el resto de caracteres el analizador nos mostrará el mensaje *LEXICAL ERROR*.

Realiza los siguientes pasos:

1. Ejecuta el scanner usando el teclado como entrada: `java Scanner` y prueba con diferentes cadenas como `print, begin, >, +, 1234`, etc. Para finalizar puedes teclear un `Ctrl-C`
2. Ejecuta de nuevo el scanner, pero ahora analizaremos el contenido del archivo *test*. Para ello vamos a utilizar el `<` para redireccionar la entrada de nuestro programa: `java Scanner < test`
3. Compara el resultado con el archivo de entrada *test*:

```
end                     SCANNER:: found Reserved Word END
+                       SCANNER:: found Operator ADD
)                       SCANNER:: found R_BRACKET
```

2.3. Modificar el archivo de entrada

Ahora nuestro objetivo es acostumbrarnos modificar el contenido de nuestro fichero analizar e identificar cada mensaje de salida con el correspondiente texto de entrada.

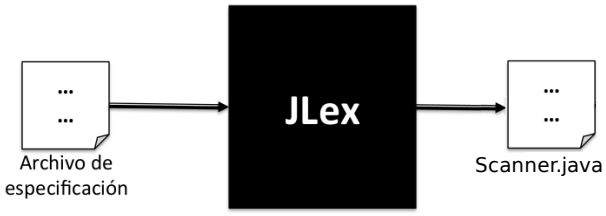
1. Recuerda que podemos editar ficheros de *ritchie* desde Windows con la aplicación WinScp. Edita el archivo *test* e introduce los siguientes cambios.
2. Añade al principio del archivo el siguiente texto: `begin123`
3. Introduce al final del archivo una línea en blanco y otra línea con el texto `print`
4. Guarda los cambios. Ejecuta el scanner: `java Scanner < test`
5. Compara los resultados obtenidos con el archivo de entrada:

```
begin123                SCANNER:: found Reserved Word BEGIN
end                     SCANNER:: found INTEGER NUMBER<123>
+                       SCANNER:: found Reserved Word END
)                       SCANNER:: found Operator ADD
                        SCANNER:: found R_BRACKET
print                   SCANNER:: found Reserved Word PRINT
```

2.4. Generar un scanner usando JLex

La tarea de programar un analizador léxico, por muy simple que sea, resulta ser realmente ardua. Es por ello que se han creado lo que se conocen como **generadores de analizadores léxicos**. Un ejemplo de éstos es **JLex**.

JLex acepta una especificación de alto nivel del conjunto de categorías sintácticas a identificar, es decir, la secuencia de caracteres a buscar en el texto de entrada. Usando el **archivo de especificación**, **JLex** genera un programa *Java* que reconoce estos patrones. ¡¡¡Este programa en Java es el analizador léxico o Scanner que deseamos crear!!!



Veamos primero la secuencia de pasos para crear un Scanner para luego explicar cómo se ejecutaría.

2.5. Pasos para crear un analizador léxico

2.5.1. Paso 1: determinar el conjunto de tokens

Claramente, primero necesitamos conocer, enumerar y describir formalmente todas y cada una de las cadenas (o tokens) que queremos reconocer: palabras reservadas, números, símbolos de puntuación, paréntesis, etc. El conjunto de tokens depende del lenguaje a desarrollar. Por ejemplo, la palabra clave **begin** o el símbolo para determinar el fin de sentencia (por ejemplo el punto y coma).

2.5.2. Paso 2: definir el patrón que describe cada tipo de cadena

Cada patrón describe un tipo de token. Por ejemplo, el patrón definido por la secuencia de caracteres **b**,**e**, **g**,**i**,**n** (en minúsculas) describe el token **begin**. Igualmente, el patrón definido con el carácter **';'** describe el símbolo de final de sentencia.

2.5.3. Paso 3: implementación

Una vez definidas los patrones para todos los tokens, se implementará el analizador utilizando **expresiones regulares**. Estas expresiones se escriben en el *archivo de especificación*, el cuál será pasado a **JLex** para ser procesado.

El formato general de un archivo fuente para **JLex** es el siguiente:

```
{subroutinas de usuario (user code)}
%%
{definiciones(JLex declarations)}
%%
{reglas (lexical rules)}
```

Es decir, se trata de tres secciones separadas por dos tantos por ciento, donde las secciones de subrutinas de usuario y de definiciones pueden ser omitidas.

La sección de reglas tiene un formato en el que cada línea representa una regla. Cada regla tiene normalmente dos partes: a la izquierda el patrón a buscar, especificado mediante una expresión regular y segunda, a la derecha, con la acción a realizar si se encuentra dicho patrón, expresado en código de Java y embebidas entre llaves.

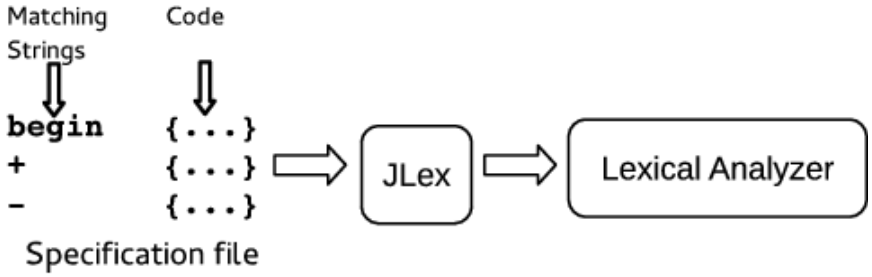
Por ejemplo, la regla para el token fin de sentencia (end of sentence, EOS) viene definida como: si se encuentra el patrón `;` entonces se imprime el texto `SCANNER:: EOS`, es decir:

```
;    { System.out.println("SCANNER:: EOS"); }
```

Por lo tanto, lo primero a realizar en este paso es describir mediante expresiones regulares todos los tokens que **JLex** debe reconocer. La segunda tarea es definir las acciones a ejecutar para cada token. Con todo ello se puede preparar el fichero de especificación del lenguaje en el formato de **JLex**.

2.5.4. Paso 4: generar el Scanner a partir del fichero de especificación

Para generar el analizador léxico usaremos **JLex** pasándole, como entrada, el fichero de especificación. La salida de **JLex** será un código en Java que implementa el analizador léxico correspondiente al archivo de especificación. Código que habría que compilar. Con el fin de facilitar esta generación del scanner, se dispone de un script denominado **generateScanner**, que se verá en los siguientes ejercicios.



3. Ejercicios

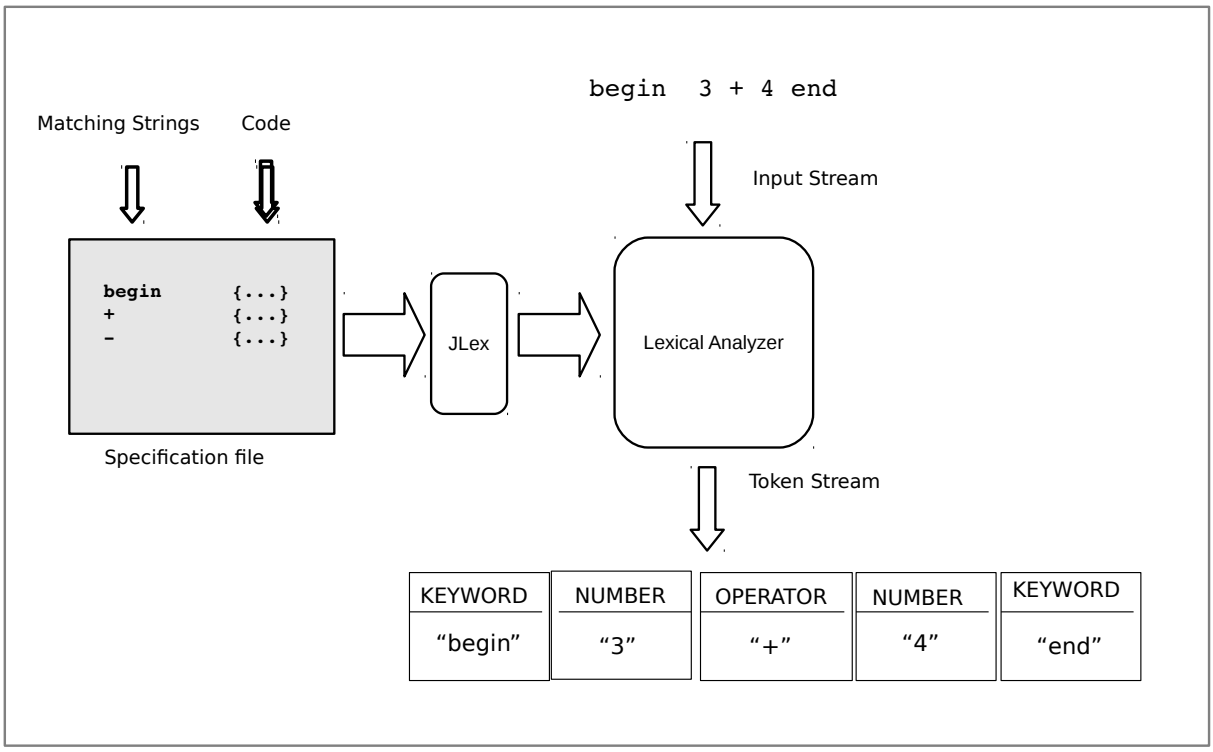
Objetivos: i) generar un Scanner, ii) identificar el fichero de especificación y el programa generado, y iii) tomar contacto con el proceso a realizar.

Realiza la siguiente secuencia de pasos:

1. Muévete al directorio **exercise2** (`cd ../exercise2`), a continuación lista y comprueba que su contenido es:
   - **specFile.lex**, un fichero de especificación que contiene las reglas de algunos patrones de texto:
     - La palabra reservada `begin`
     - Los símbolos `+` y `;`
     - Caracteres a ignorar: espacios en blanco, tabuladores y salto de línea
   - **generateScanner**, el fichero de comandos (o script) que utilizaremos para generar los scanners mediante **JLex**.
   - **test**, un fichero de prueba para el lenguaje definido en *specFile.lex*.
2. Abrir el fichero de especificación (**specFile.lex**) y descubrir las reglas, con las expresiones regulares y las acciones asociadas.
3. Generar el scanner ejecutando desde el *prompt*: `$ sh generateScanner specFile` (*¡sin la extensión!*). Lista el contenido del directorio, ¿Cuántos archivos hay ahora? ¿Cuál es el que contiene el scanner?

4. Ejecutar el analizador léxico generado

Este paso ya lo hemos realizado con anterioridad. Simplemente se trata de ejecutar el programa en Java pasando como entrada el archivo de prueba (en este caso, *test*). Luego, la orden es como antes: `java Scanner < test`. El scanner generará la salida correspondiente a las reglas definidas y según los patrones que se identifiquen en la entrada leída.



5. Ejercicios

Objetivos: i) descubrir la importancia del archivo de especificación y de cada una de sus líneas en la generación de un Scanner; ii) modificar este fichero iii) identificar algunos cambios producidos en el Scanner por modificaciones en el fichero de especificación; y iv) practicar el proceso general de análisis léxico.

1. Comprobar la correspondencia entre el fichero de test y la salida
   - Edita o muestra el contenido del fichero *test*.
   - Ejecuta el scanner sobre test: `java Scanner < test`
   - Observa la salida y relaciónala con *test*.
2. Cambiar una regla en el .lex
   - Edita el archivo de especificación **specFile.lex**, reemplaza la expresión regular *begin* por *BEGIN*.
   - Genera el scanner otra vez: `sh generateScanner specFile`
   - Ejecuta el scanner sobre *test*: `java Scanner < test`
   - Describe qué ocurre y por qué.
3. Añadir una regla al .lex
   - Edita el archivo de especificación, deshaz el cambio anterior (volvemos a tener *begin*) y añade la expresión regular para el token **end** después de la regla del *begin*. Como acción de la regla se debe imprimir el mensaje "SCANNER:: END".
   - Editar el archivo *test* y añade al inicio del mismo la cadena *beginend*.
   - Genera de nuevo el scanner: `sh generateScanner specFile`
   - Ejecuta el scanner con *test*: `java Scanner < test`
   - Interpreta lo que está ocurriendo y explica por qué ocurre.