PROCESADORES DE LENGUAJES I
PRÁCTICA DE LABORATORIO 1

Esta práctica supone la primera toma de contacto con la herramienta ANTLR (*ANother Tool for Language Recognition*). ANTLR es una herramienta que integra la generación de analizadores léxicos, sintácticos, árboles de sintaxis abstracta y evaluadores de atributos. ANTLR está escrita en Java y genera Java, C++ y C#.
En esta primera práctica no entraremos a explicar los aspectos propios del lenguaje asociado a ANTLR sino que nos centraremos en el proceso de instalación de la herramienta y ejecución de un analizador especificado en ANTLR.
Todos los detalles de esta herramienta se encuentran en su página oficial `http://www.antlr.org/`.

Un lenguaje ejemplo
Utilizaremos como lenguaje ejemplo el de las expresiones aritméticas. La entrada a procesar consistirá en una serie de expresiones aritméticas separadas por el símbolo ";". Los operadores permitidos son la suma y la resta, y se podrán utilizar paréntesis para agrupar subexpresiones. Una posible entrada a procesar sería:

```
1+1-3;
2+5+(8-2);
2+2+2;
1;
```

Para esta salida, la aplicación nos responderá con la ausencia de mensajes en el caso de que la entrada se ajuste a la descripción del lenguaje o con un mensaje de error en el caso contrario.

Especificación del analizador
ANTLR permite especificar cada analizador (léxico, sintáctico, semántico) en un fuente independiente o en un único fuente. En este primer ejemplo optaremos por utilizar un único fuente aunque a medida que los analizadores sean más complejos será recomendable especificar cada uno por separado. El analizador para nuestro lenguaje tendría un aspecto como este en ANTLR:

```
//////////////////////////////
// Analizador léxico
//////////////////////////////

class Analex extends Lexer;

BLANCO: (' '|'\t'|"\r\n") {$setType(Token.SKIP);};

NUMERO: ('0'..'9')+;

OPERADOR: '+'|'-';

PARENTESIS: '('|')';

SEPARADOR: ';';
```

```
////////////////////////////////
// Analizador sintáctico
////////////////////////////////

class Anasint extends Parser;

instrucciones : (expresion ";")* ;

expresion : exp_base (("+"|"-") exp_base)* ;

exp_base : NUMERO
         | "(" expresion ")"
         ;
```

La estructura de un fuente ANTLR

Los ficheros de gramáticas tienen la siguiente estructura:

```
header{
/* código de cabecera */
}
options{
/* opciones comunes a toda la especificación */
}

////////////////////////////////
// Definición de analizadores
////////////////////////////////
...
```

Las dos primeras secciones son opcionales. La sección `header` sirve para incluir código de cabecera, fundamentalmente instrucciones `import` o definición del paquete al que pertenecerá la clase del analizador. La sección options permite configurar algunos aspectos de ANTLR a través de opciones, que se representan mediante asignaciones del tipo `nombre_opcion = valor;`. Tras estas secciones se incluye la definición de los analizadores, ANTLR permite especificar todos los analizadores en un único fuente o dedicar un fichero independiente para cada uno.

La definición de un analizador sigue la siguiente estructura:

```
class nombre_analizador extends tipo_analizador;

options {
   /* opciones específicas del analizador */
}

tokens {
  /* definición de tokens */
}

{
 /* código propio del analizador */
}
```

```
//----------------------------
// Zona de reglas
//----------------------------
...
```

Donde:

- La primera instrucción sirve para establecer cual será el nombre del analizador y de qué clase heredará (`tipo_analizador`). En el caso de analizadores léxicos se extenderá la clase `Lexer`, para los sintácticos se usará `Parser` y `TreeParser` para los recorridos de árboles de sintaxis abstracta.
- La sección `options` será opcional y servirá para especificar opciones propias del analizador. Se puede, por ejemplo, definir el número de símbolos de anticipación en el reconocimiento LL(k), importar tokens, etc.
- La sección `tokens` también es opcional, permite definir nuevos *tokens* que se añaden a los definidos en otros analizadores.
- La zona de código nativo sirve para incluir declaraciones de atributos y métodos que se añaden a las que de forma automática se generan para la clase que implementa el analizador.
- Por último la zona de reglas constituye el núcleo de la especificación. En los ejemplos previos ya hemos mostrado el aspecto que tienen estas reglas.
- Los comentarios se pueden incluir en cualquier parte del fuente, permitiéndose comentarios de una sola línea (con el símbolo `//`) y de varias líneas (con los símbolos `/*` y `*/` para marcar el comienzo y final respectivamente).

Instalación de ANTLR

La instalación de ANTLR es bastante simple, tan sólo hay que seguir los siguientes pasos:

- Instalar un compilador de java (por ejemplo en `c:\jsdk`).
- Incluir en la variable `PATH` el directorio actual (‘`.`’) y el directorio donde está instalado el compilador de java (`c:\jsdk\bin`).
- Descargar ANTLR de `www.antlr.og`, descomprimir el fichero en una carpeta (por ejemplo `c:\antlr272`).
- Incluir en la variable `CLASSPATH` la carpeta actual (‘`.`’) y el fichero `c:\antlr272\antlr.jar`

Ejecutando el intérprete

Hasta ahora hemos especificado los distintos analizadores, pero aún nos queda por escribir un último fuente para poder ejecutarlos, el que describe la clase que contiene el método `main`:

```
////////////////////////////////
// Expre.java (clase principal)
////////////////////////////////

import java.io.*;

import antlr.collections.AST;
import antlr.ANTLRException;
```

```
public class Expre {
   public static void main(String args[]) {
      try {
         FileInputStream fis =
            new FileInputStream("entrada.txt");
         Analex analex = null;
         Anasint anasint = null;

         analex = new Analex(fis);

         anasint = new Anasint(analex);
         anasint.instrucciones();
      }catch(ANTLRException ae) {
         System.err.println(ae.getMessage());
       }catch(FileNotFoundException fnfe) {
         System.err.println("No se encontró el fichero");
       }
   }
}
```

El código es bastante claro y en general se limita a:

- Abrir el fichero `"entrada.txt"`.
- Crear un objeto de cada analizador.
- Lanzar el método `anasint.instrucciones()` para realizar el análisis sintáctico.
- Capturar las distintas excepciones que se hayan podido lanzar durante el proceso.

Una vez que tenemos todos los fuentes llega el momento de hacer la prueba definitiva. En primer lugar tendremos que crear el fichero `"entrada.txt"` e incluir en él un texto susceptible de ser reconocido por los analizadores. Por ejemplo:

```
1+1+1;
2-(2+1);
```

Como ya hemos comentado, se pueden dedicar un fichero de gramáticas para cada analizador o se pueden incluir todos en uno. Dado que el ejemplo del enunciado es extremadamente simple optaremos por incluirlos todos en un único fuente que denominaremos `Expre.g` (se suele utilizar la extensión `".g"` para los ficheros de gramáticas) y lo guardaremos en la carpeta `c:\Expre`. Una vez hecho esto ya estamos en disposición de ejecutar ANTLR, lo haremos con la siguiente orden:

```
C:\Expre>java antlr.Tool Expre.g
```

Si no hay ningún error, se generarán en la carpeta `c:\Expre` varios ficheros. Los que más nos interesan son `Analex.java` y `Anasint.java`, ya que contienen la implementación de los analizadores léxico y sintáctico respectivamente. El siguiente paso será compilar todos los ficheros java que constituyen la aplicación:

```
C:\Expre>javac *.java
```

Y ya podemos ejecutar el reconocedor y comprobar el resultado, lanzando el método `main` de la clase `Expre`:

```
C:\Expre>java Expre
```

Ante una entrada correcta no se apreciará ninguna salida, ya que no se ha especificado ninguna acción asociada al reconocimiento. Sin embargo ante una entrada incorrecta se generará un mensaje de error. Por ejemplo la entrada:

```
1++1;
```

Provocará el lanzamiento de una excepción (`ANTLRException`) que al ser capturada dará lugar a la emisión del siguiente mensaje:

```
line 1:3: unexpected token: +
```

Lectura de flujos de bytes y flujos de caracteres

Los reconocedores generados por ANTLR pueden adaptarse con facilidad para leer desde distintos flujos de entrada. La clase que implementa el análisis léxico (en nuestro ejemplo `Analex`) es la encargada de recibir ese flujo de entrada y su constructor puede recibir tanto objetos de la clase `InputStream` para recibir flujos de bytes, como de la clase `Reader` para recibir flujos de caracteres.

Como ya hemos visto en el ejemplo anterior, `FileInputStream` (una subclase de `InputStream`) nos sirve para procesar entradas almacenadas en un fichero. En el siguiente ejemplo veremos cómo `StringReader` (una subclase de `Reader`) nos permitirá aplicar el análisis léxico al contenido de una cadena de caracteres que a su vez ha sido leída desde el teclado:

```
////////////////////////////////
// Expre.java (clase principal)
////////////////////////////////

import java.io.*;

import antlr.collections.AST;
import antlr.ANTLRException;

public class Expre {
   public static void main(String args[]) {
      try {
         InputStreamReader isr =
            new InputStreamReader(System.in);
         BufferedReader br = new BufferedReader(isr);

         String linea = br.readLine();
         while (!linea.equals("$")) {
            Analex analex = null;
            Anasint anasint = null;

            analex = new Analex(new StringReader(linea));
            anasint = new Anasint(analex);
            anasint.expresion();

            linea = br.readLine();
         }
      }catch(ANTLRException ae) {
         System.err.println(ae.getMessage());
```

```
        }catch(IOException ioe) {
           System.err.println(ioe.getMessage());
        }
    }
}
```

Los aspectos más interesantes de este programa son:

- Se construye el flujo de caracteres `isr` aplicando `InputStreamReader` sobre el flujo de bytes `System.in`.
- A partir del flujo `isr` se obtiene `br`, un flujo de caracteres con buffer (`BufferedReader`) que puede leerse línea a línea.
- La entrada del teclado se lee línea a línea en la variable `linea` en un bucle que se repite hasta que se introduce `$`, el carácter elegido para indicar el fin del proceso.
- Para analizar el contenido de la variable línea basta con crear a partir de ella un flujo de entrada con el constructor `StringReader`. Dicho flujo se utilizará en la construcción del analizador léxico.

Ejercicios

1. Instalar ANTLR.
2. Compilar los fuentes presentados en el enunciado y comprobar el funcionamiento del reconocedor de expresiones propuesto.
3. Modificar la clase principal de manera que en lugar de leer siempre del fichero `"entrada.txt"` reciba el nombre del fichero a procesar a través del parámetro `args[0]` del método `main`.
4. Adaptar el reconocedor del apartado 2 de manera que la entrada se lea del teclado y no de un fichero. La entrada se procesará línea a línea y por tanto no será necesario tener en cuenta el símbolo `;` como separador. El proceso terminará cuando se introduzca la entrada `$`. Ante entradas correctas el intérprete no emitirá ningún mensaje mientras que para entradas incorrectas se mostrará el mensaje de error generado por ANTLR. Un ejemplo de sesión con este intérprete de expresiones será:

   ```
   C:\Expre>java Expre
   expre> 1 + 1
   expre> 1 +  + 1
   line 1:6: unexpected token: +
   expre> 1
   expre> $
   C:\Expre>java Expre
   ```

   ¿Se detecta bien una entrada incorrecta de la forma `2 2` ?¿Cómo se puede solucionar este problema?