Procesamiento de Lenguajes (PL)
Curso 2016/2017

Práctica 1: analizador descendente recursivo

Fecha y método de entrega

La práctica debe realizarse de forma individual, como las demás prácticas de la asignatura, y debe entregarse a través del servidor de prácticas del DLSI **antes de las 23:59 del 3 de marzo de 2017**.

Al servidor de prácticas del DLSI se puede acceder de dos maneras:

- Desde la web del DLSI (http://www.dlsi.ua.es), en el apartado "Entrega de prácticas"
- Desde la URL http://pracdlsi.dlsi.ua.es

Una vez en el servidor, se debe elegir la asignatura PL y seguir las instrucciones.

Descripción de la práctica

La práctica consiste en implementar un analizador sintáctico descendente recursivo, como se ha estudiado en la clase de teoría, utilizando Java como lenguaje fuente.

Como subprograma del analizador sintático será necesario realizar un analizador léxico, que será utilizado en la práctica 2. Además, el analizador léxico y sintáctico serán la base de la práctica 3.

Analizador léxico

El analizador léxico debe reconocer los siguientes componentes léxicos:

Token	Expresión regular	Cadena a mostrar
pari	`(`	`(`
pard	`)`	`)`
mulop	`'*'|'/'`	`* /`
addop	`'+'|'-'`	`+ -`
pyc	`;`	`;`
dosp	`:`	`:`
coma	`,`	`,`
asig	`=`	`=`
llavei	`{`	`{`
llaved	`}`	`}`
class	`class`	`'class'`
public	`public`	`'public'`
private	`private`	`'private'`
float	`float`	`'float'`
int	`int`	`'int'`
return	`return`	`'return'`
entero	`[0-9]+`	`numero entero`
id	`[a-zA-Z][a-zA-Z0-9]*`	`identificador`
real	`[0-9]+\.[0-9]+`	`numero real`

Ten en cuenta las siguientes cuestiones:

1. Debes diseñar el diagrama de transiciones en papel, tal y como se ha estudiado en clase de teoría. Para esta práctica no es necesario que pongas las palabras reservadas en el diagrama (saldrían muchos estados), puedes tratar las palabras reservadas como identificadores en el diagrama, y justo antes de devolver el token, comprobar si el identificador coincide con alguna de las palabras reservadas para devolver el token asociado en vez del token del identificador.

2. El lenguaje es sensible a mayúsculas, es decir, "`Class`" es un identificador y "`class`" es la palabra reservada `class`.

3. El analizador léxico debe ignorar los espacios en blanco, tabuladores y saltos de línea (excepto para contar columnas y filas, en cuyo caso el tabulador se contará como un único caracter). Debe leer el fichero carácter a carácter, hasta completar un token. En ningún caso debe leer el fichero completo en memoria, ni leer más de una vez el fichero.

4. En caso de que algún carácter de la entrada no pueda formar parte de un token, se debe emitir el siguiente error:

   ```
   Error lexico (f,c): caracter 'a' incorrecto
   ```

   donde `f` es la fila, `c` es la columna, y `a` es el carácter incorrecto. Cuando se encuentre un error, el programa debe terminar con `System.exit(-1)`

   Si el carácter incorrecto es el que representa el final del fichero, entonces el mensaje debe ser:

   ```
   Error lexico: fin de fichero inesperado
   ```

5. El analizador léxico también debe ignorar los comentarios, que comenzarán por "`/*`" y se extenderán (posiblemente a lo largo de varias líneas) hasta que aparezca la secuencia "`*/`". No se permiten comentarios anidados, es decir, el primer "`*/`" cierra el primer "`/*`". Si se termina el fichero en mitad de un comentario se debe emitir el error de fin de fichero inesperado.

6. En los programas que se utilizarán para probar las prácticas solamente habrá caracteres del código ASCII, es decir, no habrá acentos ni `ñ` ni otros caracteres extraños.

7. El analizador léxico se debe implementar con dos clases en Java:

   - La clase `Token`, que tendrá los tipos de token de la especificación, tal y como se ha explicado en la clase de teoría. El método `toString` debe devolver para cada token la cadena que aparece en la tercera columna de la tabla anterior.
   - La clase `AnalizadorLexico`, que debe implementar al menos el método público `siguienteToken`, que leerá caracteres de la entrada hasta formar un token y lo devolverá.

8. Utiliza la clase `RandomAccessFile` para acceder al fichero de entrada, usando el método `readByte()` o `read()` para leer caracteres. El método `readChar()` lee caracteres Unicode y no sirve para leer caracteres ASCII.

Analizador sintáctico

La gramática que debe utilizarse como base para diseñar el analizador sintáctico es la siguiente:

S	$\longrightarrow$	*C*
C	$\longrightarrow$	**class id llavei** *B V* **llaved**
B	$\longrightarrow$	**public dosp** *P*
B	$\longrightarrow$	ϵ
V	$\longrightarrow$	**private dosp** *P*
V	$\longrightarrow$	ϵ
P	$\longrightarrow$	*D P*
P	$\longrightarrow$	ϵ
D	$\longrightarrow$	*Tipo* **id pari** *Tipo* **id** *L* **pard** *Cod*
D	$\longrightarrow$	*C*
Cod	$\longrightarrow$	**pyc**
Cod	$\longrightarrow$	*Bloque*
L	$\longrightarrow$	**coma** *Tipo* **id** *L*
L	$\longrightarrow$	ϵ
Tipo	$\longrightarrow$	**int**
Tipo	$\longrightarrow$	**float**
Bloque	$\longrightarrow$	**llavei** *SecInstr* **llaved**
SecInstr	$\longrightarrow$	*Instr* **pyc** *SecInstr*
SecInstr	$\longrightarrow$	ϵ
Instr	$\longrightarrow$	*Tipo* **id**
Instr	$\longrightarrow$	**id asig** *Expr*
Instr	$\longrightarrow$	*Bloque*
Instr	$\longrightarrow$	**return** *Expr*
Expr	$\longrightarrow$	*Expr* **addop** *Term*
Expr	$\longrightarrow$	*Term*
Term	$\longrightarrow$	*Term* **mulop** *Factor*
Term	$\longrightarrow$	*Factor*
Factor	$\longrightarrow$	**real**
Factor	$\longrightarrow$	**entero**
Factor	$\longrightarrow$	**id**
Factor	$\longrightarrow$	**pari** *Expr* **pard**

Ten en cuenta las siguientes cuestiones:

1. Para probar el analizador sintáctico necesitas el analizador léxico de la práctica funcionando correctamente.
2. Calcula los conjuntos de predicción y comprueba que la gramática resultante es LL(1). Antes de ponerte a ello, debes eliminar la recursividad por la izquierda y los factores comunes por la izquierda (si aparecen).
3. En caso de que el analizador encuentre un error, se debe emitir el siguiente mensaje:

   ```
   Error sintactico (f,c): encontrado 'token', esperaba ...
   ```

   donde `f` es la fila, `c` es la columna (del primer carácter del token), y `token` es el lexema del token incorrecto, y después de `esperaba` debe aparecer una lista con los tokens que se esperaba, mostrando para cada token la cadena que aparece en la tercera columna de la tabla de los componentes léxicos. Como en el analizador léxico, si aparece algún error se debe terminar la ejecución con `System.exit(-1)`

   Ejemplo:

   ```
   Error sintactico (4,5): encontrado 'hola', esperaba ) ; + -
   ```

Los tokens esperados deben aparecer en este orden (el que tienen en la gramática): **class**, **id**, **llavei**, **llaved**, **public**, **dosp**, **private**, **pari**, **pard**, **pyc**, **coma**, **int**, **float**, **asig**, **return**, **addop**, **mulop**, **real**, **entero**.

Si el token encontrado es el final del fichero, el mensaje de error debe ser:

```
Error sintactico: encontrado fin de fichero, esperaba ...
```

4. El analizador sintáctico se debe implementar en una clase en Java, denominada `AnalizadorSintacticoDR`, que tendrá al menos los métodos/funciones asociados a los no terminales de la gramática.

5. Se publicará un script para comprobar que el analizador sintáctico funciona razonablemente bien. Se recomienda utilizar un método privado para acumular los números de regla, y un atributo booleano (flag) en la parte privada de la clase que permita mostrar o no los números de las reglas que ha ido aplicando el analizador sintáctico, de forma que se pueda reutilizar fácilmente el código para la práctica 3. Por ejemplo, ante este fichero de entrada (y suponiendo que el atributo booleano está a `true`):

```
class A {
    public:
      int f1(int n,float s);
      float ff1(float r) {
        return 2+r;
      }
    private:
      class B {
      }
}
```

la salida del analizador debería ser:

```
 1 2 3 7 9 15 15 13 16 14 11 7 9 16 16 14 12 17 18 23 24 27 31 29 25 27 32 29 26 19 8 5 7 10 2 4 6 8
```

Los números de las reglas se corresponden con la gramática obtenida eliminando la recursividad por la izquierda (RI), no con la gramática descrita anteriormente. Para que los números sean comunes (y la autocorrección automática funcione bien), las reglas nuevas resultantes de eliminar dichas características no LL(1) se colocarán en el mismo lugar que las originales, desplazando hacia abajo las siguientes reglas. Por ejemplo, al eliminar la recursividad por la izquierda de las reglas de *Expr* se reemplazarían las reglas 24 y 25 de la gramática original por 3 nuevas reglas, que tendrían los números 24, 25 y 26; por tanto, las reglas de *Term* pasarían a tener los números 27 y 28 (en vez de 26 y 27), y así sucesivamente.

6. Los números de regla se deben ir acumulando (utilizando un atributo privado de tipo `StringBuilder`, y un método privado), y cuando el análisis termine con éxito, en el método `.comprobarFinFichero()`, se imprimirán los números de regla. Si se produce algún error léxico o sintáctico, no saldrá ningún número de regla.

7. La práctica debe tener varias clases en Java:

   - La clase `plp1`, que tendrá solamente el siguiente programa principal (y los `import` necesarios, sin declarar un `package`):

```
class plp1 {
    public static void main(String[] args) {

        if (args.length == 1)
        {
          try {
            RandomAccessFile entrada = new RandomAccessFile(args[0],"r");
            AnalizadorLexico al = new AnalizadorLexico(entrada);
            AnalizadorSintacticoDR asdr = new AnalizadorSintacticoDR(al);

            asdr.S(); // simbolo inicial de la gramatica
            asdr.comprobarFinFichero();
          }
          catch (FileNotFoundException e) {
            System.out.println("Error, fichero no encontrado: " + args[0]);
          }
```

```
        }
        else System.out.println("Error, uso: java plp1 <nomfichero>");
    }
}
```

- La clase `AnalizadorSintacticoDR`, que tendrá los métodos/funciones asociados a los no terminales del analizador sintáctico, que imprimirán las reglas que van aplicando (según el valor del atributo booleano mencionado anteriormente).
- Las clases `Token` y `AnalizadorLexico` del analizador léxico