Procesamiento de Lenguajes (PL)
Curso 2018/2019

Práctica 2: analizador sintáctico SLR

Fecha y método de entrega

La práctica debe realizarse de forma individual, y debe entregarse a través del servidor de prácticas del DLSI **antes de las 23:59 del 24 de marzo de 2019**. Los ficheros fuente en Java se comprimirán en un fichero llamado "`plp2.tgz`", y no debe haber directorios (ni `package`) en él, solamente los fuentes en Java.

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 diseñar un analizador sintáctico ascendente, utilizando el algoritmo de análisis por desplazamiento-reducción, y construyendo las tablas con el método SLR. La gramática sería la siguiente:

S	$\longrightarrow$	**program id pyc** B
D	$\longrightarrow$	**var** L **endvar**
L	$\longrightarrow$	$L\ V$
L	$\longrightarrow$	V
V	$\longrightarrow$	**id dosp** *Tipo* **pyc**
Tipo	$\longrightarrow$	**integer**
Tipo	$\longrightarrow$	**real**
B	$\longrightarrow$	**begin** D SI **end**
SI	$\longrightarrow$	SI **pyc** I
SI	$\longrightarrow$	I
I	$\longrightarrow$	**id asig** E
I	$\longrightarrow$	**write pari** E **pard**
I	$\longrightarrow$	B
E	$\longrightarrow$	E **opas** F
E	$\longrightarrow$	F
F	$\longrightarrow$	**numentero**
F	$\longrightarrow$	**numreal**
F	$\longrightarrow$	**id**

Ten en cuenta las siguientes cuestiones:

1. Construye las tablas de análisis utilizando el método SLR, explicado en clase de teoría (deben salir 38 estados, del 0 al 37). Después, implementa el analizador a partir de esas tablas. Esta gramática no debes modificarla, la recursividad por la izquierda (y los factores comunes por la izquierda) no impiden construir un analizador ascendente.

2. Como en el analizador de la práctica 1, en caso de que se 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, y `token` es el carácter 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 de la práctica 1. Si aparece algún error se debe terminar la ejecución con `System.exit(-1)`

Ejemplo:

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

Los tokens esperados deben aparecer en el mismo orden que en la tabla de la práctica 1, para lo que la tabla ACCIÓN del analizador debe tener los tokens en este mismo orden: **pari**, **pard**, **dosp**, **asig**, **pyc**, **opas**, **program**, **var**, **endvar**, **integer**, **real**, **begin**, **end**, **write**, **id**, **numentero**, **numreal**.

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

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

3. La práctica debe tener varias clases en Java:
   - La clase `plp2`, que tendrá solamente el siguiente programa principal (y los `import` necesarios):

```
class plp2 {
    public static void main(String[] args) {

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

            aslr.analizar();
          }
          catch (FileNotFoundException e) {
            System.out.println("Error, fichero no encontrado: " + args[0]);
          }
        }
        else System.out.println("Error, uso: java plp2 <nomfichero>");
    }
}
```

   - La clase `AnalizadorSintacticoSLR`, que tendrá al menos el método `analizar` para analizar la cadena de entrada.
   - Las clases `Token` y `AnalizadorLexico` de la práctica 1, convenientemente adaptadas ambas al lenguaje de esta práctica. Ten en cuenta que desaparecen los tokens `coma`, `cori`, `cord`, `ptopto`, `opmul`, `array`, `of` y `pointer`.

4. La salida de la práctica debe ser los números de las reglas que ha ido aplicando el analizador sintáctico. Por ejemplo, ante este fichero de entrada:

```
program prueba;

begin
 var a : integer;
     b : real;
     c : integer;
 endvar

   a := 7 - b + 2 + 2.3;
   write(c + 2- 3 - 4.5 );
   begin
     var e:real; endvar
     e := 7
   end
end
```

la salida de la práctica debería ser:

```
1 8 9 13 8 10 11 15 16 2 4 5 7 9 12 14 17 14 16 14 16 15 18 10 11 14 17 14 16 14 18 15 16 2 3 5 6 3 5 7 4 5 6
```

Nota: como el analizador SLR obtiene la inversa de una derivación por la derecha de la cadena de entrada, es necesario almacenar las reglas por las que se reduce en un vector y luego imprimir el vector en orden inverso para que salga una derivación correcta[1].

[1] También se puede utilizar una pila para almacenar los números de regla, que facilita que los números salgan en orden inverso.