Se desea construir un analizador sintáctico en JAVA, basado en la técnica LL(1), para reconocer el lenguaje de las listas de números del lenguaje Prolog. Estas listas se componen de una secuencia, separada por comas, de átomos, (en este caso números), o bien de otras listas, y encerradas entre paréntesis cuadrados. Los siguientes son ejemplos de secuencias que pertenecen a este lenguaje:

```
[]
[1]
[[]]
[[],[]]
[[1],[2,3]]
[[[5],[12,4]],6,7]
[[3],[[4,[5]],[]],[6,[],8]]
```

No pertenecen al lenguaje elementos aislados, secuencias incompletas o secuencias en las que no encajen los paréntesis.

Se pide:

Apartado A (5 puntos)
Diseñar un programa en JAVA que tenga como entrada un fichero con una lista y de cómo resultado CORRECTO en caso de que la lista este bien formada (pertenezca al lenguaje) y ERROR en caso contrario.

Apartado B (2 puntos, sólo si el apartado A es correcto)
Extender el lenguaje para que reconozca entradas de la forma `nElem( <lista> )`, en cuyo caso el programa deberá devolver `NELEM=<número de elementos>`, es decir debe contar el número de elementos que hay en la lista de entrada; o ERROR en caso de que la lista no este correctamente formada. La siguiente tabla muestra algunos ejemplos:

Entrada	Salida
`nElem( [[1],[2,3]] )`	`NELEM=3`
`nElem( [] )`	`NELEM=0`
`nElem( [[],[2]] )`	`NELEM=1`
`nElem( [[3],[[4,[5]],[]],[6,[],8]] )`	`NELEM=5`

Apartado C (2 puntos, sólo si el apartado A es correcto)
Extender el lenguaje para que reconozca entradas de la forma `maxDepth( <lista> )`, en cuyo caso el programa deberá devolver `MAXDEPTH=<profundidad máxima>`, es decir debe analizar la lista y obtener el valor máximo de anidamiento de las listas; o ERROR en caso de que la lista no este correctamente formada. La siguiente tabla muestra algunos ejemplos:

Entrada	Salida
`maxDeptrh( [[1],[2,3]] )`	`MAXDEPTH=2`
`maxDeptrh( [] )`	`MAXDEPTH=1`
`maxDeptrh( [[],[[3,4],5]]] )`	`MAXDEPTH=3`
`maxDeptrh( [[3],[[4,[5]],[]],[6,[],8]] )`	`MAXDEPTH=4`

Apartado D (1 punto, sólo si el apartado A es correcto)

Extender el lenguaje para que reconozca entradas de la forma `maxLength( <lista> )`, en cuyo caso el programa deberá devolver `MAXLENGTH=<longitud máxima>`, es decir debe analizar la lista y obtener el valor máximo del número de elementos de la lista o cualquiera de sus sublistas; o ERROR en caso de que la lista no este correctamente formada. La siguiente tabla muestra algunos ejemplos:

Entrada	Salida
`maxLength( [[1],[2,3,4]] )`	`MAXLENGTH=3`
`maxLength( [] )`	`MAXLENGTH=0`
`maxLength( [[],[[3,4],5]]] )`	`MAXLENGTH=2`
`maxLength([[3],[[4,[5]],[]],[6,[],8]] )`	`MAXLENGTH=3`

NOTAS:

Para completar las especificaciones y responder a cualquier duda sobre el funcionamiento del programa que se pide construir, se proporciona un programa de ejemplo ya compilado que realiza exactamente las funciones que se piden, y un conjunto de casos de prueba. Este programa se encuentra en un fichero comprimido para distintos sistemas operativos. En Linux, una vez descomprimido será necesario modificar los permisos de ejecución mediante la instrucción:

```
chmod +x parser
```

La ejecución del programa, se realiza mediante la instrucción:

```
./parser <fichero de entrada>
```

Para facilitar la construcción del analizador se proporciona el analizador léxico denominado `Lex.lex` para JFLEX, que devuelve los tokens representados como números enteros. Para obtener el analizador léxico debe ejecutarse la instrucción:

```
jflex Lex.lex
```

También se proporciona un esquema de la clase que implementará el analizador sintáctico denominado `Parser.java` que debe completarse con la implementación necesaria de manera que pueda compilarse y ejecutar con la siguiente secuencia de instrucciones:

```
javac Parser.java
java Parser <fichero de entrada>
```

obteniendo la misma salida que se obtendría con el programa de ejemplo.

Solo debe enviarse el fichero `Parser.java` por lo que cualquier modificación de cualquier otro fichero no tendrá efecto.