MATRICES -1

Implementar mediante *JFlex* y *CUP* un interprete que realice cálculos básicos con matrices de números reales. El siguiente es un ejemplo de programa en este lenguaje:

```
a = [1, 2, 3 ; 6, 5, 4 ; 8, 7, 9];
print(a);
b = transpuesta(a);
print(b);
c = [1,2,3,4; 5,6,7,8];
print(c+c);
f = c * transpuesta(c);
print(inversa(f));
```

Las matrices son siempre de dos dimensiones *n* x *m* , siendo $1<n<1000$ & $1<m<1000$.[1] Ciertas operaciones, como la inversa solo se pueden realizar sobre matrices cuadradas, es decir $n=m$. Las operaciones aritméticas entre matrices requieren a su vez condiciones conocidas: Por ejemplo la suma de matrices requiere que ambas matrices tengan las mismas dimensiones, el producto requiere que el número columnas de la primera matriz coincida con el numero de filas de la segunda.

Se proporciona un analizador léxico implementado mediante *JFlex*, en el fichero `Matrices.flex`,; la clase `Matrices.java` que contiene la implementación de las operaciones entre matrices[2] y el método `main`; y una tabla de símbolos auxiliar en la clase `TablaSimbolos.java`. La lista de operaciones con matrices disponibles se limita a los definidos en el analizador léxico.

Para compilar y ejecutar este interprete se usarán la secuencia:

```
cup Matrices.cup
jflex Matrices.flex
javac Matrices.java
java Matrices <entrada>
```

Para la corrección de este ejercicio se entrega SOLAMENTE el fichero `Matrices.cup` por lo que cualquier modificación en cualquier otro fichero no se tendrá en cuenta al corregir.

Se proporcionan diversos casos de prueba[3], con diversos aspectos que deben controlarse en la implementación.

Caso 1. Declaración e impresión de matrices

Entrada	Salida		
`print( [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]) ;`	1,00	2,00	3,00
	4,00	5,00	6,00
	7,00	8,00	9,00

Caso 2. Llamada a funciones

Entrada	Salida		
`print( transpuesta([1,2,3; 4,5,6; 7,8,9]) );`	1,00	4,00	7,00
	2,00	5,00	8,00
	3,00	6,00	9,00

[1] No se acepta la matriz vacía []. La determinación del tamaño exacto de la matriz se hará en tiempo de ejecución

[2] Para realizar la salida debe emplearse necesariamente el método `print(double[][])` de la clase `Matrices` pero para la lectura puede utilizarse la clase `ArrayList<ArrayList<Double>>`. Se proporcionan métodos de conversión entre ambas representaciones.

[3] Los casos de prueba que se muestran son solo un ejemplo, a los que llamaremos casos de prueba públicos. Para evitar implementaciones "ad hoc" se probaran otros casos de prueba similares a estos, pero que no se proporcionan. A estos casos se les llama casos de prueba privados. Para superar cada prueba hay que superar ambos conjuntos de casos de prueba.

Caso 3. Operaciones

Entrada	Salida
`print( [1,2,3; 4,5,6] * [1,2; 3,4; 5,6]);`	`22,00 28,00` `49,00 64,00`
`print( [1,2,3; 4,5,6] + [9,8,7; 6,5,4]);`	`10,00 10,00 10,00` `10,00 10,00 10,00`
`print( [1,2;3,4]+[5,6,7;7,6,5]*[1,2;3,4;5,6]);`	`59,00 78,00` `53,00 72,00`

Caso 4. Comprobación de dimensiones correctas en una matriz

Entrada	Salida
`print( [1,2,3; 4,5; 7,8,9] );`	`ERROR1 ...`
`print( [] );`	`Syntax error ...`

Caso 5. Comprobación de dimensiones correctas en funciones y operaciones

Entrada	Salida
`print( inversa([1,2,3; 4,5,6]) );`	`ERROR2 ...`
`print( [1,2,3; 4,5,6] * [1,2,3; 4,5,6]);`	`ERROR4 ...`

Caso 6. Definición y uso de variables

Entrada	Salida

`A = [1,2,3; 4,5,6];`
`B = [1,2; 3,4; 5,6];`
`print(A*B);`
	`22,00 28,00`
	`49,00 64,00`

Caso 7. Expresiones y operaciones complejas

Entrada	Salida

`a = [1,2,3,4];`
`b = transpuesta(a)*a;`
`print(b);`
	`1,00 2,00 3,00 4,00`
	`2,00 4,00 6,00 8,00`
	`3,00 6,00 9,00 12,00`
	`4,00 8,00 12,00 16,00`

`c = [1,2,3,4; 7,3,2,1; 4,3,9,5];`
`a = [1,0,3,4; 4,0,2,1; 7,4,3,2];`
`f = a * (transpuesta(c)+transpuesta(a))`
`  + ([1;5;2]+transpuesta([2,4,8]))*[3,6,9];`
`print(f*inversa(f));`
	`1,00 0,00 0,00`
	`0,00 1,00 0,00`
	`0,00 0,00 1,00`

Caso 8. Aceptar también matrices en formato Java

Entrada	Salida

`A = [1,2,3; 4,5,6];`
`B = {{1,2},{3,4},{5,6}};`
`print(A*B);`
	`22,00 28,00`
	`49,00 64,00`