Implementar mediante JFlex y CUP un interprete que realice cálculos básicos con vectores y conjuntos de números reales (positivos y negativos). El siguiente es un ejemplo de programa en este lenguaje:

```
a = {1,2,3};
b = set {3,4,5};
c = a : b;
d = a + b:3:4;
print(d);
f = set {1,1,2,3,4,5,-6}
print(f & set c);
```

Las variables una vez declaradas, pueden usarse pero no pueden modificarse. La única diferencia entre vectores y conjuntos es que los conjuntos no tienen elementos repetidos. Se acepta expresamente el vector y el conjunto vacío que se escriben como:

```
v = {};
c = set {};
```

El operador unario `set`, que se aplica a conjuntos o a vectores y los transforma en un conjunto. A pesar de ser unario tiene mayor prioridad que cualquier otro operador.

Se definen los operadores binarios entre números, vectores y conjuntos, todos ellos asociativos por la izquierda: (ver ejemplos en los casos de prueba)

Se introducen los siguientes operadores entre vectores:

- Concatenación: (Operador :) Añade los elementos de un vector al final de otro. Este operador tiene la máxima prioridad. Puede concatenar dos vectores o bien un vector y un número. Este operador tiene mayor prioridad que la suma.
- Suma: (Operador +) Suma uno a uno los elementos de un vector. Las dimensiones de ambos vectores deben ser iguales

Se introducen los siguientes operadores binarios entre conjuntos:

- Unión: (Operador :) Construye un conjunto con los elementos de los elementos comunes a dos conjuntos. Al igual que en el caso de los vectores, también se puede añadir un número a un conjunto. (Es lo mismo que concatenar dos vectores y eliminar los elementos duplicados).
- Intersección: (Operador &) Devuelve los elementos comunes a dos conjuntos. La intersección es el operador binario de mayor prioridad.

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 vectores y conjuntos, métodos auxiliares[1] y el método `main`. La lista de funciones 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[2], con diversos aspectos que deben controlarse en la implementación.

---

[1] Para realizar la salida debe emplearse necesariamente los métodos `print(double[])` de la clase `Matrices`, tanto para vectores como para conjuntos. Puede también utilizarse las clases `ArrayList<Doble>` para la construcción del vector. Se proporcionan métodos de conversión entre ambas representaciones, asi como métodos para convertir vectores en conjuntos y viceversa.

[2] 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 1. Declaración e impresión de vectores y conjuntos. (`zp1a.mat`, `zp1b.mat`, etc.)

Entrada	*Salida*
`b = {1,2,3};` `print(b);`	`1,00 2,00 3,00`
`b = set {1,2,2,3,3};` `print(b);`	`1,00 2,00 3,00`

`a = set {1,2,2,3,3};`
`b = set a;`
`print(set b);`
	`1,00 2,00 3,00`

`a = set {-1,1,-2,-2,3,3};` `print(set a);`	`-2,00 -1,00 1,00 3,00`

Caso 2. Operaciones con vectores. Concatenación y suma. (`zp2a.mat, zp2b.mat`, etc.)

Entrada	*Salida*

`b = {1,2,3};`
`c = {4,5};`
`print(b:c);`
	`1,00 2,00 3,00 4,00 5,00`

`b = {1,2}:{3,4}:{5,6};` `print(b);`	`1,00 2,00 3,00 4,00 5,00 6,00`
`b = {1,2}:3:{4} + 4:{3,2,1};` `print(b);`	`5,00 5,00 5,00 5,00`
`a = 1:{2,3,4}+{4,5,6}:7;` `print(a);`	`5,00 7,00 9,00 11,00`

Caso 3. Operaciones con conjuntos. Unión (Concatenación) de conjuntos. (`zp3a.mat`, `zp3b.mat`, etc.)

Entrada	*Salida*

`a = set {1,2,3};`
`b = set {3,4,5};`
`print(a:b);`
	`1,00 2,00 3,00 4,00 5,00`

`a = set {1,2};`
`b = a:a:a;`
`print(b);`
	`1,00 2,00`

`a = {1,2,3};`
`b = {2,3,4};`
`print(set a: set b);`
	`1,00 2,00 3,00 4,00`

Caso 4. Operaciones con conjuntos. Intersección (`zp4a.mat`, `zp4b.mat`, etc.)

Entrada	*Salida*

`a = set {1,2,3};`
`b = set {3,4,5};`
`print(a&b);`
	`3,00`

`a = set {1,2,3};`
`b = set {2,4,6};`
`c = a & set {2,3,4,5,6} & b;`
`print(c);`
	`2,00`

`a = set {1,2,3};`
`b = a & set {2,3,4} : set {2,7};`
`print(b);`
	`2,00 3,00 7,00`

Caso 5. Interoperabilidad entre conjuntos y vectores. Los vectores y los conjuntos son interoperables. La concatenación (o unión) de un vector y un conjunto, (o de un conjunto y un vector) da como resultado un vector. La unión de dos conjuntos da como resultado un conjunto. (`zp5a.mat`, `zp5b.mat`, etc.). Para alterar la prioridad de operadores se pueden usar los paréntesis.

Entrada	*Salida*

`a = {1,2,3};`
`b = set {2,3,4};`
`c = a:b;`
`print(c);`
	`1,00 2,00 3,00 2,00 3,00 4,00`

`a = {1,2,3};`
`b = set {2,3,4};`
`c = set a : b;`
`print(c);`
	`1,00 2,00 3,00 4,00`

`a = {1,2,3};`
`b = set {2,3,4} : a;`
`c = a : set b;`
`print (a:set(b:set c));`
	`1,00 2,00 3,00 1,00 2,00 3,00 4,00`

Caso 6. Conversión de tipos implícita. La intersección solo puede realizarse entre conjuntos, por lo que en caso necesario se realizará una conversión implícita de vector a conjunto. (`zp6a.mat`, `zp6b.mat`, etc.). Para alterar la prioridad de operadores se pueden usar los paréntesis.

Entrada	*Salida*
`a = {1,2,3} & {2,3,4};` `print(a);`	`2,00 3,00`
`a = {1,2} : {3} & set {1,3,4};` `print(a);`	`1,00 2,00 3,00`
`a = set {1,2} & {1,3} : {1,3,4};` `print(a);`	`1,00 3,00 4,00`
`a = {1,2} & (set ({1,3} : {1,3,4}));` print(a);	`1,00`

ANEXO:

En el fichero `Matrices.java` puede encontrar, entre otros muchos, métodos para:

La representación interna tanto de vectores como de conjuntos puede hacerse usando los tipos `double[]` o bien `ArrayList<Double>`. Se proporcionan métodos para convertir entre una representación y otra:

Convertir un `ArrayList<Double>` en un vector de tipo `double[]`

```
public static double[] toVector(ArrayList<Double> arrayList)
```

Convertir un vector de tipo `double[]` en un `ArrayList<Double>`

```
public static ArrayList<Double> toArrayList(double[] vector)
```

Para convertir un vector en un conjunto solo es necesario eliminar los elementos duplicados usando alguno de estos dos métiodos:

Convertir un `ArrayList<Double>` en un conjunto de tipo `ArrayList<Double>`

```
public static ArrayList<Double> toSet(ArrayList<Double> arrayList)
```

Convertir un tipo `double[]` en un conjunto de tipo `double[]`

```
public static double[] toSet(double[] vector)
```

Concatenar un escalar y un vector

```
public static double[] concatena(double n, double[] vector)
```

Concatenar un vector y un escalar

```
public static double[] concatena(double[] vector, double n)
```

Concatenar dos vectores (como `double[]`)

```
public static double[] concatena(double[] vector1, double[] vector2)
```

Concatenar dos vectores (como `ArrayList<Double>`)

```
public static ArrayList<Double> concatena(ArrayList<Double> vector1,
                                          ArrayList<Double> vector2)
```

Sumar dos vectores

```
public static double[] suma(double[] vector1, double[] vector2)
```

Unir un escalar y un conjunto para dar un conjunto

```
public static double[] union(double numero, double[] vector )
```

Unir un conjunto y un escalar para dar un conjunto

```
public static double[] union(double[] vector, double numero)
```

Unir dos conjuntos (como `double[]`) para dar un conjunto

```
public static double[] union(double[] vector1, double[] vector2)
```

Unir dos conjuntos (como `ArrayList<Double>`) para dar un conjunto

```
public static ArrayList<Double> union(ArrayList<Double> vector1,
                                      ArrayList<Double> vector2)
```

Intersectar dos conjuntos (como `double[]`) para dar un conjunto

```
public static double[] interseccion(double[] vector1, double[] vector2)
```

Intersectar dos conjuntos (como `ArrayList<Double>`) para dar un conjunto

```
public static ArrayList<Double> interseccion(ArrayList<Double> set1,
                                             ArrayList<Double> set2)
```

Imprimir un número, un vector o un conjunto, (dependiendo de su representación interna)

```
public static void print(double n)
public static void print(double[] vector)
public static void print(ArrayList arrayList)
```

Constante para la salida del error de dimensiones diferentes de vectores

```
public final static String ERROR_SUMA_VEC ;
```