



## MATRICES -3

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, o bien un vector y un conjunto. Este operador tiene mayor prioridad que la suma. El operador `:` aplicado a un conjunto y a un vector da como resultado un vector.
- 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. Solo si se concatenan dos conjuntos el resultado es otro conjunto, ya que al concatenar un vector con un conjunto o un conjunto con un vector el resultado es un vector.
- Intersección: (Operador `&`) Devuelve los elementos comunes a dos conjuntos. La intersección es el operador binario de mayor prioridad. La intersección de un vector con un conjunto, o de un conjunto con un vector da como resultado un vector.

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<sup>1</sup> 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<sup>2</sup>, con diversos aspectos que deben controlarse en la implementación.

---

<sup>1</sup> 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, así como métodos para convertir vectores en conjuntos y viceversa.

<sup>2</sup> 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 probarán 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.

<i>Entrada</i>	<i>Salida</i>
<code>a = {1,2,3}; b = set {2,3,4}; c = a:b; print(c);</code>	1,00 2,00 3,00 2,00 3,00 4,00
<code>a = {1,2,3}; b = set {2,3,4}; c = set a : b; print(c);</code>	1,00 2,00 3,00 4,00
<code>a = {1,2,3}; b = set {2,3,4} : a; c = a : set b; print (a:set(b:set c));</code>	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.

<i>Entrada</i>	<i>Salida</i>
<code>a = {1,2,3} &amp; {2,3,4}; print(a);</code>	2,00 3,00
<code>a = {1,2} : {3} &amp; set {1,3,4}; print(a);</code>	1,00 2,00 3,00
<code>a = set {1,2} &amp; {1,3} : {1,3,4}; print(a);</code>	1,00 1,00 3,00 4,00
<code>a = {1,2} &amp; (set ({1,3} : {1,3,4})); print(a);</code>	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 <code>double[]</code> o bien <code>ArrayList&lt;Double&gt;</code> . Se proporcionan métodos para convertir entre una representación y otra: Convertir un <code>ArrayList&lt;Double&gt;</code> en un vector de tipo <code>double[]</code> <b><code>public static double[] toVector(ArrayList&lt;Double&gt; arrayList)</code></b> Convertir un vector de tipo <code>double[]</code> en un <code>ArrayList&lt;Double&gt;</code> <b><code>public static ArrayList&lt;Double&gt; toArrayList(double[] vector)</code></b>
Para convertir un vector en un conjunto solo es necesario eliminar los elementos duplicados usando alguno de estos dos métodos: Convertir un <code>ArrayList&lt;Double&gt;</code> en un conjunto de tipo <code>ArrayList&lt;Double&gt;</code> <b><code>public static ArrayList&lt;Double&gt; toSet(ArrayList&lt;Double&gt; arrayList)</code></b> Convertir un tipo <code>double[]</code> en un conjunto de tipo <code>double[]</code> <b><code>public static double[] toSet(double[] vector)</code></b>
Concatenar un escalar y un vector <b><code>public static double[] concatena(double n, double[] vector)</code></b> Concatenar un vector y un escalar <b><code>public static double[] concatena(double[] vector, double n)</code></b> Concatenar dos vectores (como <code>double[]</code> ) <b><code>public static double[] concatena(double[] vector1, double[] vector2)</code></b> Concatenar dos vectores (como <code>ArrayList&lt;Double&gt;</code> ) <b><code>public static ArrayList&lt;Double&gt; concatena(ArrayList&lt;Double&gt; vector1, ArrayList&lt;Double&gt; vector2)</code></b>
Sumar dos vectores <b><code>public static double[] suma(double[] vector1, double[] vector2)</code></b>
Unir un escalar y un conjunto para dar un conjunto <b><code>public static double[] union(double numero, double[] vector)</code></b> Unir un conjunto y un escalar para dar un conjunto <b><code>public static double[] union(double[] vector, double numero)</code></b> Unir dos conjuntos (como <code>double[]</code> ) para dar un conjunto <b><code>public static double[] union(double[] vector1, double[] vector2)</code></b> Unir dos conjuntos (como <code>ArrayList&lt;Double&gt;</code> ) para dar un conjunto <b><code>public static ArrayList&lt;Double&gt; union(ArrayList&lt;Double&gt; vector1, ArrayList&lt;Double&gt; vector2)</code></b>
Interseccionar dos conjuntos (como <code>double[]</code> ) para dar un conjunto <b><code>public static double[] interseccion(double[] vector1, double[] vector2)</code></b> Interseccionar dos conjuntos (como <code>ArrayList&lt;Double&gt;</code> ) para dar un conjunto <b><code>public static ArrayList&lt;Double&gt; interseccion(ArrayList&lt;Double&gt; set1, ArrayList&lt;Double&gt; set2)</code></b>
Imprimir un número, un vector o un conjunto, (dependiendo de su representación interna) <b><code>public static void print(double n)</code></b> <b><code>public static void print(double[] vector)</code></b> <b><code>public static void print(ArrayList arrayList)</code></b>
Constante para la salida del error de dimensiones diferentes de vectores



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