Programación II
Práctica 0: Acumuladores booleanos

Como vimos en clase un **acumulador** booleano toma sólo dos valores, true o false.
Se denomina acumulador, porque suma un resultado parcial al resultado final de la función.
La variable "ret" es el nombre abreviado de "valor de retorno".

Cuantificadores

Cuando queremos probar una propiedad P para todo un conjunto de datos:

$$\{\forall\ x\ \in lista\ /\ P(x)\} \equiv true$$

Utilizaremos la hipótesis de ret = ***true*** y la acumulación será de la forma:

```
ret = ret AND P(x)
```

Cuando queremos probar una propiedad P para un solo elemento:

$$\{\exists\ x\ \in lista\ /\ P(x)\} \equiv true$$

Utilizaremos la hipótesis de ret = ***false*** y la acumulación será de la forma:

```
ret = ret OR P(x)
```

Ejercicios obligatorios:

1) Implementar con acumuladores una función booleana "mayor10" que recibe una lista de números, que sea verdadera si todos los números son mayores a 10.

```
boolean mayor10(int[] lista){ …  }
```

2) Implementar una función que determine si un arreglo es subconjunto de otro:

```
public static boolean pertenecenTodos(int[] elems,
                                      int[] arreglo) ...
```

Casos borde a tener en cuenta:

- *elems* está vacío (y la función devuelve verdadero)
- *arreglo* está vacío (y la función devuelve falso)
- alguno de los arreglos contiene duplicados (no influye, es suficiente con que estén una vez)

Algunos ejemplos:

```
[1, 2] ⊆ [3, 2, 1]
[4, 1] ⊈ [1, 2, 3]
```

```
[2, 2] ⊆ [1, 2, 3]
```

3) Implementar la función, utilizando acumuladores booleanos, que reciba una matriz de enteros, y devuelva verdadero ⇔ en cada una de las filas, existe al menos un número negativo.

```
public static boolean tieneNegativos(int[][] mat){ ...
```

4) Implementar una función que, dada una matriz de enteros, verifique que:

   a) todas las filas están en orden estrictamente ascendente
   b) todas las columnas tienen al menos un elemento impar, y otro par

Signatura y documentación:

```
// Pre-condición: "int[][] mtx" es una matriz N × M, esto es:
// todas las filas tienen longitud N y todas las columnas, M;
// con N, M > 0.
//
// No es necesario verificar explícitamente la pre-condición:
// de no cumplirse, el código puede devolver cualquier valor, o
// lanzar una excepción (p.ej. ArrayIndexOutOfBoundsException).


public static boolean mayorDiversidad(int[][] mtx) ...
```

Algunos ejemplos:

```
[[1, 2, 3], [4, 5, 6]] → Verdadero
[[1, 2, 3], [4, 5, 5]] → Falso
[[1, 2, 3], [2, 4, 6]] → Falso

No cumplen la pre-condición:

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

5) Implementar funciones con potencias de 2 y logaritmos en base 2

   a) Implementar una función que dado un vector de enteros devuelva verdadero ⇔ todos sus elementos son potencia de 2.

Algunos ejemplos:

```
[8, 2, 32] → Verdadero [ 2³ ,  2¹  ,  2⁵ ]
[15, 2, 8] → Falso.   No son todos potencias de 2
```

b) Utilizar el punto 5) para implementar una función que dada una matriz de enteros devuelva verdadero ⇔ en alguna fila todos sus elementos son potencia de 2.

Algunos ejemplos:

```
[[1, 2, 3], [8, 2, 1]] → Verdadero    fila 2 = [ 2³, 2¹, 2⁰ ]
[[1, 2, 3], [11, 2, 8], [[4, 5, 6]] → Falso ninguna lo cumple
```

c) Implementar una función que dada una matriz de enteros devuelva verdadero ⇔ en alguna fila algún elemento es **Parte Entera($\log_2(c+1)$)** donde c es el índice de la columna.

Algunos ejemplos:

```
[[1, 2, 3], [0, 1, 1]] → Verdadero    fila 1 lo cumple
                    Parte entera(log₂1)=0
                    Parte entera(log₂2)=1
                    Parte entera(log₂3)=1
[[1, 2, 3], [11, 2, 8], [[4, 5, 6]] → Falso ninguna lo cumple
```

d) **IMPORTANTE**: Interpretar cual es la relación entre 2^n y $\log_2(n)$

Ejercicio de parcial

Implementar una función que dada una matriz de N x N elementos enteros y un arreglo de N elementos enteros determine si el elemento i del arreglo se encuentra en la fila i de la matriz:

```
public static boolean arregloEnFilas(int[][] mat,
                                      int[] arreglo) ...
```

Casos borde a tener en cuenta:

- *arreglo* está vacío (y la función devuelve verdadero)
- *mat* está vacío (y la función devuelve falso)
- la matriz contiene duplicados (no influye, es suficiente con que estén una vez)

9	2	**4**	5
6	**7**	1	7
3	**5**	9	11
12	8	5	**1**

4
7
5
1

Devuelve True

4	2	13	5
6	7	1	3
3	7	9	11
12	8	5	10

4
7
5
1

Devuelve False