Calcular el primo anterior y posterior

Un número entero **n**, mayor que 1, es primo si tiene solamente por divisores la unidad y el propio número. Una forma común de comprobar si un número es primo consiste en comprobar todos los posibles divisores (desde 2 a **n-1**).

En este ejercicio se solicita realizar dos funciones:

A) Realizar una función que llamaremos **esPrimo**. Recibirá como parámetro un número entero largo (**num**), y devolverá a través de su valor de retorno un valor 1 si el **num** recibido es primo y un 0 en caso contrario.

Previamente a realizar cualquier cálculo, la función deberá validar el parámetro **num** recibido. Si éste fuera 0 o negativo, devolverá directamente un valor **-**1 como valor de retorno. Si fuera 1 lo considerará no primo y devolverá directamente un valor 0.

En caso de superar las validaciones anteriores, procederá a realizar el cálculo y a devolver por el valor de retorno el indicador correspondiente, según determine que es primo o no.

B) Realizar una función que llamaremos **primosContiguos** que trabajará con tres parámetros enteros largos, uno de entrada (**num**) y dos de salida (**anterior** y **posterior**). Además, la función devolverá un valor de retorno, según se explica más adelante.

Con respecto al parámetro de entrada, **num**, la función deberá validar primeramente que dicho parámetro es mayor que 1, en caso contrario devuelve **-**1.

En caso de superar las validaciones anteriores, la función deberá calcular:

- El número primo inmediatamente superior al valor **num** recibido, y lo devolverá a través del parámetro de salida **posterior**.
- El número primo inmediatamente anterior y mayor que 1 al valor **num** recibido, y lo devolverá a través del parámetro de salida **anterior**. Si no existiera, devolverá en el parámetro **anterior** el valor **-**1.
- Devolverá 0 a través del valor de retorno.

Para realizar los cálculos indicados, es obligatorio que esta función llame a la anterior función **esPrimo**.

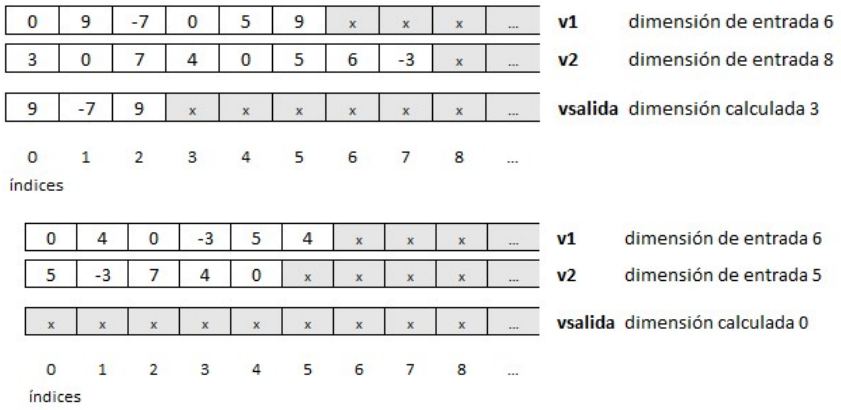
Copiar vector elementos no incluidos

Se solicita crear una función llamada **copiaNoIncluidos**. La función definirá los parámetros adecuados para recibir dos vectores de entrada (**v1** y **v2**) de números enteros, y para devolver en salida un tercer vector (**vsalida**) también de números enteros.

La función deberá copiar en **vsalida**, todos los elementos de **v1** que no se encuentren en **v2**. Los elementos deberán ser copiados en el mismo orden en que aparecen en **v1**.
La dimensión del vector **vsalida**, deberá ser calculada dentro de la función dependiendo del número de elementos que se copien en este vector, y será devuelta a través del parámetro correspondiente.

La función no tiene valor de retorno.

Se muestran varios ejemplos de comportamiento de la función.



Ejemplos de copia en vsalida, de elementos de v1 no existentes en v2

B) Supongamos que tenemos en el programa principal las siguientes definiciones:

```
#define DIM 50

int vector1[DIM];
int dimv1;          // Dimensión efectiva de vector1
int vector2[DIM];
int dimv2;          // Dimensión efectiva de vector2

int vectorcopia[DIM];
int dimvcopia;

.../...
// Se suponen cargadas las variables vector1 y vector2, y sus dimensiones
// dimv1 y dimv2, que representan el número de elementos efectivos de los dos
// vectores

 --- INDICAR CÓMO SERÍA AQUÍ LA INVOCACIÓN A LA FUNCIÓN copiaNoIncluidos ---
.../...
.../...
```

Matriz marco perfecto

Se dice que una matriz, que no tiene por qué ser cuadrada, de dimensión superior a 3 x 3, es una **matriz marco** cuando la suma de los elementos de

- su primera fila +
- su última fila +
- su primera columna +
- su última columna

es mayor que la suma de sus elementos interiores (no incluidos en las filas y columnas anteriores).

Una matriz marco, se dice que además es una **matriz marco perfecto** si además, todas las columnas interiores (sin considerar la primera fila, última fila, primera columna y última columna) suman lo mismo. Para que una matriz sea marco perfecto, tiene que cumplir que se una matriz marco.

A) Escribir una función que llamaremos **matrizMarco**. La función definirá los parámetros adecuados para recibir una matriz, y cuya dimensión física de columnas está definida en el módulo llamador mediante la constante "define" COL.
La función determinará si es una **matriz marco** y si además es una **matriz marco perfecta** y devolverá el resultado de su decisión a través de un parámetro de salida que tendrá el siguiente valor…

- **0** si la matriz **no** es una **matriz marco**
- **1** si la matriz es una **matriz marco** pero no es marco perfecto
- **2** si la matriz es una **matriz marco perfecto**

La función no tiene valor de retorno.

Se presentan varios ejemplos que aclaran la definición hecha.

1	3	5	-2	4	8
1	5	8	9	9	7
-3	0	9	8	0	1
-2	0	-5	0	10	15
1	1	-5	0	4	0

Suma elementos "marco" 39
Suma elementos interiores 53
La matriz NO es un marco

1	3	5	-2	4	8
1	3	2	12	1	7
-3	-1	4	-3	15	1
-2	7	3	0	-7	15
1	1	-5	0	4	0
	9	9	9	9	**suma columnas**

Suma elementos "marco" 39
Suma elementos interiores 36
La matriz es un marco perfecto

1	3	5	-2	4	8
1	3	2	12	1	7
-3	-1	4	-3	15	1
-2	6	2	0	-8	15
1	1	-5	0	4	0
	8	8	**9**	8	**suma columnas**

Suma elementos "marco" 39
Suma elementos interiores 33
La matriz es un marco (pero no es marco perfecto)

B) Supongamos que tenemos en el programa principal las siguientes definiciones:

```
#define FIL 50
#define COL 50

int matrizA[FIL][COL];
int filas, columnas; // Dimensiones efectivas (filas, columnas) de matrizA

int tipoMarco;       // Indicador de si es matriz marco y tipo

.../...
// Se supone cargada la variable matrizA y su dimensión filas y columnas, que
// representa el número de filas y columnas de matrizA


   --- INDICAR CÓMO SERÍA AQUÍ LA INVOCACIÓN A LA FUNCIÓN matrizMarco ---

.../...
.../...
```