MÓDULO 1. INTRODUCCIÓN A LA PROGRAMACIÓN

Ejercicios Adicionales RP7

1. Diseña una función `numOcurrencias` que reciba como parámetros dos arrays de enteros (`numeros` y `permutacion`) y devuelva el número de ocurrencias de cualquier permutación (mismos elementos aunque puedan estar en orden diferente) de los elementos del array `permutacion` en el array `números`.

   Para calcular las ocurrencias del array `permutacion` en el array `numeros` se hace lo siguiente (la parte sombreada del array `numeros` es lo que se comprueba en cada paso):

   1) Se "coloca" el array `permutacion` sobre el array `numeros` en la posición 0 de éste y se comprueba si los elementos de `permutacion` son una permutación de la parte del array `numeros` sobre la que está colocado el array `permutacion`. Por ejemplo:

1	4	1	12

   `permutacion` **Sí hay ocurrencia**

12	1	1	4	1	7	14	1	12	12	4

   `numeros`

   2) Se "desplaza" el array `permutacion` a la posición 1 sobre el array `numeros` y se hace la misma comprobación.

1	4	1	12

   `permutacion` **NO hay ocurrencia**

12	1	1	4	1	7	14	1	12	12	4

   `numeros`

   3) Se repite el mismo proceso mientras sea posible "colocar" el array `permutacion` sobre el array `numeros`.

permutacion

1	4	1	12

No hay ocurrencia

12	1	1	4	1	7	14	1	12	12	4

numeros

Para este ejemplo la función devolvería el valor 1, ya que hay una sola ocurrencia.

Crea también una función **`main`** para leer de teclado dos colecciones de números enteros (terminadas en 0), almacenarlas en dos arrays, invocar a la función anterior y mostrar por pantalla el número de ocurrencias que devuelve la misma.

NOTAS:

- Podrá haber números repetidos en ambos arrays.
- Los elementos contenidos en los arrays serán siempre naturales **mayores que 0**.
- Pueden usarse arrays auxiliares si se considera necesario.

2. Una cadena de caracteres puede estar formada por diferentes subcadenas separadas por determinados caracteres separadores. Por ejemplo la cadena:

"este-aunque sencillo-es un ejemplo"

está formada por las subcadenas "este", "aunque", "sencillo", "es", "un", "ejemplo" si consideramos como separadores el espacio (' ') y el guión ('-'). A cada una de estas subcadenas se le denomina también *token*.

Diseña una función `obtenerTokens` que reciba dos parámetros:

- `cadena`, que contiene la cadena de caracteres de la que hay que obtener los tokens. Podemos suponer que no aparecerá más de un separador de forma consecutiva en `cadena`. Esto es, los tokens estarán separados por un solo separador, aunque éste pueda ser diferente a lo largo de la cadena.
- `separadores`, que es una cadena de caracteres que contiene los posibles separadores a considerar a la hora de realizar la obtención de los tokens.

La función devolverá un array con los tokens obtenidos de `cadena` considerando los separadores que aparecen en `separadores`. Este array no tendrá elementos repetidos.

Crea también una función **`main`** para leer de teclado dos cadenas de caracteres (`cadena` y `separadores`), invocar a la función anterior y mostrar por pantalla el contenido del array que devuelve la misma.

3. Un número natural n se dice que es triangular cuando su valor se puede obtener mediante la suma de unos cuantos números naturales consecutivos, comenzando con el número 1. Por ejemplo, el número 15 es triangular porque 15 = 1+2+3+4+5.

   El juego de cartas llamado "El solitario Búlgaro" consiste en lo siguiente. Consideremos un mazo de *n* cartas, siendo *n* un número triangular. Se reparte la totalidad de las cartas en un número arbitrario de montones, cada uno de ellos con una cantidad arbitraria de cartas. El lote de montones se puede reorganizar así:

   Se toma una carta de cada montón y con todas ellas se forma un nuevo montón, que se agrega al final de la lista de montones. Los montones que tenían una única carta desaparecen.

   Si repetimos esta operación con los montones que van quedando, se asegura que finalmente obtendremos una disposición tal que el primer montón tendrá 1 carta, el segundo 2 cartas, el tercero 3 cartas y así sucesivamente hasta el último.

   Por ejemplo, partiendo de un mazo de 15 cartas repartidas en tres montones con 5, 7 y 3 cartas cada uno, las reorganizaciones sucesivas evolucionan como sigue:

   [5 7 3]

   [4 6 2 3]

   [3 5 1 2 4]

   [2 4 1 3 5]

   [1 3 2 4 5]

   [2 1 3 4 5]

   [1 2 3 4 5]

   Diseña un algoritmo (la función *main* y los procedimientos y funciones que sean necesarios) que lea de teclado el número de montones inicial y el número de cartas de cada montón. El algoritmo llevará a cabo el proceso descrito anteriormente e irá mostrando por pantalla el contenido de los montones en cada paso hasta finalizar el solitario (ver ejemplo anterior).

   Nota: Podemos suponer que el número de cartas del que se parte es un número triangular.

4. Diseña un algoritmo (la función *main* y los procedimientos y funciones que sean necesarios) que lea de teclado una cadena de caracteres (*patron*), un número natural *x* y un texto, y muestre por pantalla un listado de todas las palabras del texto que contengan al menos *x* caracteres en común con la cadena *patron*.

   Ejemplo:

   Entrada:

```
Introduzca el patrón: ALICIA

Introduzca el valor de x: 2

Introduzca el texto (FIN para terminar):

CREO QUE IREMOS A LA DIRECCION QUE NOS DIERON Y TAMBIEN
IREMOS A LA OTRA DIRECCION QUE YO CONOCIA FIN
```

Salida:

```
LA DIRECCION TAMBIEN LA DIRECCION CONOCIA
```

Ya que LA contiene la L y la A, DIRECCION contiene la C y la I, TAMBIEN contiene la A y la I y CONOCIA la A y la I

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- El carácter separador de palabras es el espacio en blanco.