8.6 Ejercicios



Ejercicios 1-14

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario. Observa bien lo que hace cada función ya que, si las implementas en el orden adecuado, te puedes ahorrar mucho trabajo. Por ejemplo, la función `esCapicua` resulta trivial teniendo `voltea` y la función `siguientePrimo` también es muy fácil de implementar teniendo `esPrimo`.

1. **esCapicua**: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.
2. **esPrimo**: Devuelve verdadero si el número que se pasa como parámetro es primo y falso en caso contrario.
3. **siguientePrimo**: Devuelve el menor primo que es mayor al número que se pasa como parámetro.
4. **potencia**: Dada una base y un exponente devuelve la potencia.
5. **digitos**: Cuenta el número de dígitos de un número entero.
6. **voltea**: Le da la vuelta a un número.
7. **digitoN**: Devuelve el dígito que está en la posición n de un número entero. Se empieza contando por el 0 y de izquierda a derecha.
8. **posicionDeDigito**: Da la posición de la primera ocurrencia de un dígito dentro de un número entero. Si no se encuentra, devuelve -1.
9. **quitaPorDetras**: Le quita a un número n dígitos por detrás (por la derecha).
10. **quitaPorDelante**: Le quita a un número n dígitos por delante (por la izquierda).
11. **pegaPorDetras**: Añade un dígito a un número por detrás.
12. **pegaPorDelante**: Añade un dígito a un número por delante.
13. **trozoDeNumero**: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.
14. **juntaNumeros**: Pega dos números para formar uno.



Ejercicio 15

Muestra los números primos que hay entre 1 y 1000.



Ejercicio 16

Muestra los números capicúa que hay entre 1 y 99999.



Ejercicio 17

Escribe un programa que pase de binario a decimal.



Ejercicio 18

Escribe un programa que pase de decimal a binario.



Ejercicio 19

Une y amplía los dos programas anteriores de tal forma que se permita convertir un número entre cualquiera de las siguientes bases: decimal, binario, hexadecimal y octal.



Ejercicios 20-28

Crea una biblioteca de funciones para arrays (de una dimensión) de números enteros que contenga las siguientes funciones:

1. **generaArrayInt**: Genera un array de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. **minimoArrayInt**: Devuelve el mínimo del array que se pasa como parámetro.
3. **maximoArrayInt**: Devuelve el máximo del array que se pasa como parámetro.
4. **mediaArrayInt**: Devuelve la media del array que se pasa como parámetro.
5. **estaEnArrayInt**: Dice si un número está o no dentro de un array.
6. **posicionEnArray**: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra.
7. **volteaArrayInt**: Le da la vuelta a un array.
8. **rotaDerechaArrayInt**: Rota n posiciones a la derecha los números de un array.
9. **rotaIzquierdaArrayInt**: Rota n posiciones a la izquierda los números de un array.



Ejercicio 29-34

Crea una biblioteca de funciones para arrays bidimensionales (de dos dimensiones) de números enteros que contenga las siguientes funciones:

1. **generaArrayBiInt**: Genera un array de tamaño n x m con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. **filaDeArrayBiInt**: Devuelve la fila i-ésima del array que se pasa como parámetro.
3. **columnaDeArrayBiInt**: Devuelve la columna j-ésima del array que se pasa como parámetro.
4. **coordenadasEnArrayBiInt**: Devuelve la fila y la columna (en un array con dos elementos) de la primera ocurrencia de un número dentro de un array bidimensional. Si el número no se encuentra en el array, la función devuelve el array {-1, -1}.
5. **esPuntoDeSilla**: Dice si un número es o no punto de silla, es decir, mínimo en su fila y máximo en su columna.
6. **diagonal**: Devuelve un array que contiene una de las diagonales del array bidimensional que se pasa como parámetro. Se pasan como parámetros fila, columna y dirección. La fila y la columna determinan el número que marcará las dos posibles diagonales dentro del array. La dirección es una cadena de caracteres que puede ser “nose” o “neso”. La cadena “nose” indica que se elige la diagonal que va del noroeste hacia el sureste, mientras que la cadena “neso” indica que se elige la diagonal que va del noreste hacia el suroeste.



Ejercicio 35

Crea una función con la siguiente cabecera:

```
public static String convierteEnPalotes(int n)
```

Esta función convierte el número n al sistema de palotes y lo devuelve en una cadena de caracteres. Por ejemplo, el 470213 en decimal es el | | | | - | | | | | | - - | | - | - | | | en el sistema de palotes. Utiliza esta función en un programa para comprobar que funciona bien. Desde la función no se debe mostrar nada por pantalla, solo se debe usar `print` desde el programa principal.



Ejercicio 36

Crea la función de manejo de arrays que tenga la siguiente cabecera y que haga lo que se especifica en los comentarios (puedes incluirla en tu propia biblioteca de rutinas):

```
public static int[] filtraPrimos(int x[]) // Devuelve un array con todos los
                                          // números primos que se encuentren
                                          // en otro array que se pasa como
                                          // parámetro.
                                          // Obviamente el tamaño del array
                                          // que se devuelve será menor o
                                          // igual al que se pasa como
                                          // parámetro.
```

Utiliza esta función en un programa para comprobar que funcionan bien. Para que el ejercicio resulte más fácil, las repeticiones de primos se conservan; es decir, si en el array x el número 13 se repite 3 veces, en el array devuelto también estará repetido 3 veces. Si no existe ningún número primo en x, se devuelve un array con el número -1 como único elemento.



Ejercicio 37

Crea una función con la siguiente cabecera:

```
public String convierteEnMorse(int n)
```

Esta función convierte el número n al sistema Morse y lo devuelve en una cadena de caracteres. Por ejemplo, el 213 es el . . _ _ _ . _ _ _ _ _ . . . _ _ en Morse. Utiliza esta función en un programa para comprobar que funciona bien. Desde la función no se debe mostrar nada por pantalla, solo se debe usar print desde el programa principal.

```
1 . _ _ _ _      6 _ . . . .
2 . . _ _ _      7 _ _ . . .
3 . . . _ _      8 _ _ _ . .
4 . . . . _      9 _ _ _ _ .
5 . . . . .      0 _ _ _ _ _
```



Ejercicio 38

Crea la función de manejo de arrays que tenga la siguiente cabecera y que haga lo que se especifica en los comentarios (puedes incluirla en tu propia biblioteca de rutinas):

```
public int[] filtraCapicuas(int x[]) // Devuelve un array con todos los números
                                     // capicúa que se encuentren en otro array
                                     // que se pasa como parámetro.
                                     // Obviamente el tamaño del array que se
                                     // devuelve será menor o igual al que se
                                     // pasa como parámetro.
```

Utiliza esta función en un programa para comprobar que funcionan bien. Para que el ejercicio resulte más fácil, las repeticiones de números capicúa se conservan; es decir, si en el array x el número 505 se repite 3 veces, en el array devuelto también estará repetido 3 veces. Si no existe ningún número capicúa en x, se devuelve un array con el número -1 como único elemento.



Ejercicio 39

Crea una función con la siguiente cabecera:

```
public String convierteEnPalabras(int n)
```

Esta función convierte los dígitos del número n en las correspondientes palabras y lo devuelve todo en una cadena de caracteres. Por ejemplo, el 470213 convertido a palabras sería:

```
cuatro, siete, cero, dos, uno, tres
```

Utiliza esta función en un programa para comprobar que funciona bien. Desde la función no se debe mostrar nada por pantalla, solo se debe usar `print` desde el programa principal. Fíjate que hay una coma detrás de cada palabra salvo al final.



Ejercicio 40

Crea la función de manejo de arrays que tenga la siguiente cabecera y que haga lo que se especifica en los comentarios (puedes incluirla en tu propia biblioteca de rutinas):

```
public int[] filtraCon7(int x[]) // Devuelve un array con todos los números
                                 // que contienen el 7 (por ej. 7, 27, 782)
                                 // que se encuentren en otro array que se
                                 // pasa como parámetro. El tamaño del array
                                 // que se devuelve será menor o igual al
                                 // que se pasa como parámetro.
```

Utiliza esta función en un programa para comprobar que funcionan bien. Para que el ejercicio resulte más fácil, las repeticiones de números que contienen 7 se conservan; es decir, si en el array x el número 875 se repite 3 veces, en el array devuelto también estará repetido 3 veces. Si no existe ningún número que contiene 7 en el array x, se devuelve un array con el número -1 como único elemento.



Ejercicio 41

Realiza un programa que pinte un triángulo relleno tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Este ejercicio ya se realizó en el tema de bucles, ahora se trata de usar una función para que la implementación sea más sencilla. Por ejemplo, se puede crear una función `linea(char caracter, int repeticiones)` que pinte una línea con el carácter especificado.

Ejemplo 1:

```
Introduzca la altura de la figura: 8
********
*******
******
*****
****
***
**
*
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5
*****
****
```

```
***
**
*
```



Ejercicio 42

Realiza un programa que pinte un triángulo hueco tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Utiliza funciones para pintar las líneas.

Ejemplo 1:

```
Introduzca la altura de la figura: 8
********
*     *
*    *
*   *
*  *
* *
**
*
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5
*****
*  *
* *
**
*
```



Ejercicio 43

Realiza un programa que pinte un triángulo relleno tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Utiliza funciones para pintar las líneas.

Ejemplo 1:

```
Introduzca la altura de la figura: 8
********
 *******
  ******
```

```
  *****
   ****
    ***
     **
      *
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5
*****
 ****
  ***
   **
    *
```



Ejercicio 44

Realiza un programa que pinte un triángulo hueco tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Utiliza funciones para pintar las líneas.

Ejemplo 1:

```
Introduzca la altura de la figura: 8
********
 *     *
  *    *
   *   *
    *  *
     * *
      **
       *
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5
*****
 *  *
  * *
   **
    *
```



Ejercicio 45

Realiza un programa que pinte un valle tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Podemos suponer que el usuario introduce una altura mayor o igual a 3.

Ejemplo 1:

```
Introduzca la altura de la figura: 8

       *             *
       **           **
       ***         ***
       ****       ****
       *****     *****
       ******   ******
       ******* *******
       ***************
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5

       *       *
       **     **
       ***   ***
       **** ****
       *********
```

Ejemplo 3:

```
Introduzca la altura de la figura: 3

       *   *
       ** **
       *****
```



Ejercicio 46

Realiza un programa que pinte un valle tal como se muestra en los ejemplos. El usuario debe introducir la altura de la figura. Podemos suponer que el usuario introduce una altura mayor o igual a 3.

Ejemplo 1:

```
Introduzca la altura de la figura: 8

       *             *
       **           **
       * *         * *
       *  *       *  *
       *   *     *   *
```

```
*    *   *    *
*     * *     *
***************
```

Ejemplo 2:

```
Introduzca la altura de la figura: 5

*       *
**     **
* *   * *
*  * *  *
*********
```

Ejemplo 3:

```
Introduzca la altura de la figura: 3

*   *
** **
*****
```



Ejercicio 47

Define la función **convierteArrayEnString** con la siguiente cabecera:

```
public static String convierteArrayEnString(int[] a)
```

Esta función toma como parámetro un array que contiene números y devuelve una cadena de caracteres con esos números. Por ejemplo, si a = { }, convierteArrayEnString(a) devuelve “”; si a = { 8 }, convierteArrayEnString(a) devuelve “8”; si a = { 6, 2, 5, 0, 1 }, convierteArrayEnString(a) devuelve “62501”.



Ejercicio 48

Define la función **concatena** con la siguiente cabecera:

```
public static int[] concatena(int[] a, int[] b)
```

Esta función toma dos arrays como parámetros y devuelve un array que es el resultado de concatenar ambos. Por ej. si a = { 8, 9, 0 } y b = { 1, 2, 3 }, concatena(a, b) devuelve { 8, 9, 0, 1, 2, 3 }.



Ejercicio 49

Escribe un programa que genere los n primeros términos de la sucesión *look and say*. El primer término es 1. A continuación se va leyendo - un uno - por tanto tenemos 11, se sigue leyendo – dos unos – por tanto tenemos 21, etc. Se recomienda usar arrays para almacenar los dígitos porque los tipos `int` y `long` son muy limitados en cuanto al número de dígitos. También puede resultar de ayuda utilizar las funciones `convierteArrayEnString` y `concatena` definidas en los ejercicios anteriores.

Ejemplo 1:

```
¿Cuántos términos de la sucesión look and say quiere calcular? 8
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211
```

Ejemplo 2:

```
¿Cuántos términos de la sucesión look and say quiere calcular? 5
1, 11, 21, 1211, 111221
```

Ejemplo 3:

```
¿Cuántos términos de la sucesión look and say quiere calcular? 12
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, 31131211131221, 13211311123113112211, 1\
1131221133112132113212221, 3113112221232112111312211312113211
```



Ejercicio 50

Define la función **mezcla** con la siguiente cabecera:

```
public static int[] mezcla(int[] a, int[] b)
```

Esta función toma dos arrays como parámetros y devuelve un array que es el resultado de mezclar los números de ambos de forma alterna, se coge un número de a, luego de b, luego de a, etc. Los arrays a y b pueden tener longitudes diferentes; por tanto, si se terminan los números de un array se terminan de coger todos los que quedan del otro.

Ejemplos:

```
Si a = {8, 9, 0} y b = {1, 2, 3}, mezcla(a, b) devuelve {8, 1, 9, 2, 0, 3 }
Si a = {4, 3} y b = {7, 8, 9, 10}, mezcla(a, b) devuelve {4, 7, 3, 8, 9, 10}
Si a = {8, 9, 0, 3} y b = {1}, mezcla(a, b) devuelve {8, 1, 9, 0, 3}
Si a = { } y b = {1, 2, 3}, mezcla(a, b) devuelve {1, 2, 3}
```



Ejercicio 51

Realiza un programa que rellene un array con 10 números aleatorios comprendidos entre 2 y 100 (ambos incluidos) y que los muestre por pantalla. A continuación, el programa indicará para cada uno de ellos si es un número primo y/o un capicúa de la forma que muestra el ejemplo.

Ejemplos:

```
Array generado:
19 22 57 11 3 52 32 46 2 14
El 19 es primo y no es capicúa.
El 22 no es primo y es capicúa.
El 57 no es primo y no es capicúa.
El 11 es primo y es capicúa.
El 3 es primo y es capicúa.
El 52 no es primo y no es capicúa.
El 32 no es primo y no es capicúa.
El 46 no es primo y no es capicúa.
El 2 es primo y es capicúa.
14 no es primo y no es capicúa.
```



Ejercicio 52

Implementa la función **aleatorioDeArray** con la cabecera que se muestra a continuación:

```
public static int aleatorioDeArray(int[] a)
```

Esta función debe devolver un número del array escogido al azar entre todos los disponibles. Por ejemplo, si a = {111, 222, 333, 444}, aleatorioDeArray(a) podría devolver el 111, el 222, el 333 o el 444. Si b = {52, 37}, aleatorioDeArray(b) podría devolver el 52 o el 37. Utiliza la función en un programa de prueba.



Ejercicio 53

Implementa una función con nombre **nEsimo** que busque el número que hay dentro de un array bidimensional en la posición n-ésima contando de izquierda a derecha y de arriba abajo, como si se estuviera leyendo. El primer elemento es el 0. Si la posición donde se busca no existe en el array, la función debe devolver -1. Se debe entregar tanto el código de la función como el código de prueba que la usa. La cabecera de la función es la siguiente:

```
public static int nEsimo(int[][] n, int posicion)
```

Si el array a es el que se muestra a continuación:

```
35 72 24 45 42 60
32 42 64 23 41 39
98 45 94 11 18 48
12 34 56 78 90 12

nEsimo(a, 0) devuelve 35
nEsimo(a, 2) devuelve 24
nEsimo(a, 5) devuelve 60
nEsimo(a, 6) devuelve 32
nEsimo(a, 21) devuelve 78
nEsimo(a, 23) devuelve 12
nEsimo(a, 24) devuelve -1
nEsimo(a, 100) devuelve -1
```



Ejercicio 54

Crea las funciones cuyas cabeceras se muestran a continuación, observa que tienen el mismo nombre:

```
public static int ocurrencias(int digito, int n)
public static int ocurrencias(int digito, int[] a)
```

La función **ocurrencias** devuelve el número de veces que aparece un dígito dentro de un número (primera función) o bien el número de veces que aparece un dígito en una serie de números contenidos en un array (segunda función).

Ejemplos:

`console ocurrencias(8, 4672) devuelve 0 ocurrencias(5, 25153) devuelve 2 ocurrencias(2, 123456) devuelve 1 Si a = {714, 81, 9, 11}, ocurrencias(1, a) devuelve 4 Si a = {42, 13, 12345, 4}, ocurrencias(4, a) devuelve 3 Si a = {6, 66, 666}, ocurrencias(6, a) devuelve 6` console

Utiliza estas funciones en un programa para comprobar que funcionan bien.



Ejercicio 55

Realiza una función que tome como parámetro un array de cadenas de caracteres y que devuelva otro array con los mismos valores habiendo eliminado las posibles repeticiones. Se distinguen mayúsculas de minúsculas, por tanto "hola" es distinto de "Hola". Por ejemplo, si el array **a** contiene los valores **{"casa", "coche", "sol", "mesa", "mesa", "coche", "ordenador", "sol", "CASA"}**, la sentencia **sinRepetir(a)** devolvería el array **{"casa", "coche", "sol", "mesa", "ordenador", "CASA"}**. Se debe entregar tanto el código de la función como el código de prueba que la usa. La cabecera de la función es la siguiente:

```
public static String[] sinRepetir(String[] s)
```



Ejercicio 56

Implementa una función con nombre **corteza** que sea capaz de extraer la capa exterior de un array bidimensional. Esta capa se extrae en forma de array de una dimensión. La extracción de números comienza en la esquina superior izquierda y continúa en el sentido de las agujas del reloj. Se debe entregar tanto el código de la función como el código de prueba que la usa. La cabecera de la función es la siguiente:

```
public static int[] corteza(int[][] n)
```

Por ejemplo, si el array bidimensional **a** es el que se muestra a continuación:

```
45 92 14 20 25 78
35 72 24 45 42 60
32 42 64 23 41 39
98 45 94 11 18 48
```

El array unidimensional generado por **corteza(a)** sería el siguiente:

```
45 92 14 20 25 78 60 39 48 18 11 94 45 98 32 35
```