Relación de ejercicios tema 1: Arrays, cadenas estilo C y matrices

Metodología de la programación, 2022-2023

Contenido:

1 Problemas básicos

1. Haz un programa que lea de la entrada estándar el número de elementos que tendrá un array de enteros, y dos enteros para el rango (mínimo, máximo) de posibles valores que tomarán los elementos del array. El programa rellenará el array con **números aleatorios** enteros en el rango proporcionado, y posteriormente lo mostrará en la salida estándar. El programa contendrá las siguientes funciones:

   - La función `main` para probar que todo funciona de forma correcta.
   - La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
   - La función `generarArray` que recibe un array de enteros, el número de elementos a usar y un rango de valores (mínimo, máximo) y lo rellena con números aleatorios en el rango proporcionado.

   A continuación se muestra un ejemplo de ejecución de este programa:

```
> ejercicio1
Número de elementos: 10
Mínimo: 0
Máximo: 100

10
98 11 3 32 21 68 4 67 82 33
```

   Si redireccionamos la salida estándar a un fichero, el array generado se puede guardar en un fichero para usarlo posteriormente en otros programas:

```
> ejercicio1 > datos1.txt
Número de elementos: 10
Mínimo: 0
Máximo: 100
```

El contenido del fichero `datos.txt` sería:

```
10
98 11 3 32 21 68 4 67 82 33
```

Nota sobre generación de números pseudoaleatorios: Para inicializar aleatoriamente un array con valores enteros entre -20 y 30, por ejemplo, puede emplearse el siguiente fragmento de código:

```
#include <iostream>
#include <cstdlib>  // rand
#include <ctime> // time
#include <cmath> // floor

/**
 * @brief Genera un valor del intervalo [minimo,maximo] ambos inclusive
 * @param minimo el mínimo valor posible a generar aleatoriamente
 * @param maximo el máximo valor posible a generar aleatoriamente
 * @return Un número entero aleatorio en el intervalo [minimo,maximo] ambos inclusive
 */
int uniforme(int minimo, int maximo)
{
   double u01= std::rand() / (RAND_MAX+1.0); // Uniforme01
   return floor(minimo + u01 * (maximo-minimo+1));
}


int main(int argc, char* argv[])
{
   int minimo = -20; // Queremos valores en el rango -30<=n<=50
   int maximo = 30;

   // Inicializa el generador con el reloj del sistema
   srand ((int) time(0));

   for (int i=0; i<100; i++){
      std::cout <<  uniforme(minimo, maximo) << " ";
   }
   std::cout << std::endl;
}
```

Se puede obtener más información sobre srand(), rand() y time() en la siguiente dirección: `http://www.cplusplus.com`.

2. Haz un programa que lea el número de elementos (como máximo 100) para un array y un conjunto de números enteros y los guarde en un array e indique cuál es el **menor de**

todos. Debe incluirse un programa principal (**main**) para probar las funciones implementadas. El programa contendrá las siguientes funciones:

- La función `main` para probar que todo funciona de forma correcta.
- La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
- La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos útiles finalmente leídos (este último parámetro se pasa por referencia).
- La función `buscarMinimoArray` que reciba un array de enteros, y dos enteros (índices inicial y final del array donde buscar el mínimo) y devuelva la posición donde se encuentra el mínimo elemento del subarray que comienza en la posición inicial y acaba en la posición final (ambas inclusive).

A continuación se muestra un ejemplo de ejecución de este programa:

```
> ejercicio2
Número de elementos: 10

Elemento 0: 98
Elemento 1: 11
Elemento 2: 3
Elemento 3: 32
Elemento 4: 21
Elemento 5: 68
Elemento 6: 4
Elemento 7: 67
Elemento 8: 82
Elemento 9: 33

Array leído: 98 11 3 32 21 68 4 67 82 33
Mínimo: 3
```

También podríamos ejecutar este programa redireccionando la entrada estándar a partir del fichero `datos1.txt` que se generó en el ejercicio 1, y así no hay que meter manualmente los datos mediante el teclado.

```
> ejercicio2 < datos1.txt

Número de elementos: 10

Elemento 0: 98
Elemento 1: 11
Elemento 2: 3
Elemento 3: 32
Elemento 4: 21
Elemento 5: 68
Elemento 6: 4
Elemento 7: 67
```

```
Elemento 8: 82
Elemento 9: 33

Array leído: 98 11 3 32 21 68 4 67 82 33
Mínimo: 3
```

3. Haz un programa que lea el número de elementos (como máximo 100) para un array y un conjunto de números enteros y los guarde en un array. Luego debe pedir que se introduzca un número entero y mostrará la posición donde el número entero se encuentra en el anterior array (o -1 si no está). El algoritmo consiste en una **búsqueda secuencial**. El programa contendrá las siguientes funciones:

   - La función `main` para probar que todo funciona de forma correcta.
   - La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
   - La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos finalmente leídos (este último parámetro se pasa por referencia).
   - La función `busquedaSecuencialArray` que reciba un dato entero a buscar, un array de enteros, y dos enteros (índices inicial y final del array donde buscar el dato) y devuelva la posición donde el número entero se encuentra en el anterior array (o -1 si no está).

4. Haz un programa que lea el número de elementos (como máximo 100) para un array y un conjunto de números enteros y los guarde en un array y lo **invierta**, mostrando el resultado. Si la secuencia introducida es **1, 100, 34, 48, 53**, entonces el resultado final mostrado debe ser **53, 48, 34, 100, 1**. El programa contendrá las siguientes funciones:

   - La función `main` para probar que todo funciona de forma correcta.
   - La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
   - La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos finalmente leídos (este último parámetro se pasa por referencia).
   - La función `invertirArray` que reciba un array de enteros y el número de elementos útiles del array, e invierta el array de entrada (sin usar ningún otro array auxiliar).

5. Haz un programa que lea el número de elementos (como máximo 100) para un array y un conjunto de números enteros y los guarde en un array. Luego debe ordenar el array con el **algoritmo de ordenación por selección** y mostrará el array ya ordenado en la salida estándar. Este algoritmo consiste en recorrer el array a partir de la posición 0, repitiendo sucesivamente los siguientes pasos:

   (a) Buscar el elemento más pequeño del array (desde posición actual hasta el final).

(b) Intercambiar el elemento más pequeño con el elemento de la posición actual.

Fíjate, que puedes usar el método `buscarMinimoArray` implementado en un ejercicio anterior para hacer el paso a).

El programa contendrá las siguientes funciones:

- La función `main`.
- La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
- La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos finalmente leídos (este último parámetro se pasa por referencia).
- La función `buscarMinimoArray` que reciba un array de enteros, y dos enteros (índices inicial y final del array donde buscar el mínimo) y devuelva la posición donde se encuentra el mínimo elemento del subarray que comienza en la posición inicial y acaba en la posición final (ambas inclusive).
- La función `ordenarporInsercion` que reciba un array de enteros y lo ordene de menor a mayor con el método de selección.

6. Haz un programa que lea el número de elementos (como máximo 100) para un array y un conjunto de números enteros ordenados de menor a mayor y los guarde en un array. Luego debe pedir que se introduzca un número entero y mostrará la posición donde el número entero se encuentra en el anterior array (o -1 si no está). El algoritmo a seguir será el de **búsqueda binaria**. Este algoritmo consiste en localizar el elemento central; si el buscado es más pequeño seguimos buscando en la primera mitad, si no, seguimos buscando en la segunda mitad. La implementación más habitual mantiene dos índices (`izq` y `der`) y usa un bucle que itera hasta que se encuentra el elemento o los índices se cruzan:

```
int izq = 0, der = n-1;

while (izq<=der) {
   int centro= posición de elemento central;
   if elemento del centro > buscado
      der = recalcular posición derecha;
   else if elemento del centro < buscado
      izq = recalcular posición izquierda;
   else
      devolver centro;
}
devolver -1;
```

El programa contendrá las siguientes funciones:

- La función `main`.

- La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
- La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos finalmente leídos (este último parámetro se pasa por referencia).
- La función `busquedaBinariaArray` que reciba un dato entero a buscar, un array de enteros, y dos enteros (índices inicial y final del array donde buscar el dato) y devuelva la posición donde el número entero se encuentra en el anterior array (o -1 si no está).

7. Haz un programa que lea una serie de números enteros (como máximo 100) y que **elimine los elementos repetidos** guardándolos en un nuevo array.

   El algoritmo empleado para eliminar los elementos repetidos consistirá en recorrer sucesivamente el array a partir de la posición 1, buscando si el elemento actual está en el subarray que comienza en la posición 0 y acaba en la posición anterior a la actual. Si es así, lo borraremos desplazando hacia la izquierda todos los elementos siguientes al elemento actual. Como puede verse, podemos hacer uso de la función `busquedaSecuencialArray` que hicimos en el ejercicio anterior.

   El programa contendrá las siguientes funciones:

   - La función `main` para probar que todo funciona de forma correcta.
   - La función `imprimirArray` para mostrar en la salida estándar el número de elementos (y un salto de línea a continuación) y los elementos de un array de enteros (separados por espacios en blanco).
   - La función `leerArray` para leer de la entrada estándar los elementos de un array. Para ello recibe como parámetro un array de enteros, su dimensión y el número de elementos finalmente leídos (este último parámetro se pasa por referencia).
   - La función `eliminarRepetidosArray` para eliminar los elementos repetidos de un array. Recibe el array de entrada, su número de elementos útiles, el array de salida y el número de elementos útiles del array de salida (este último por referencia).

   Haz una segunda versión de la función (`eliminarRepetidosArray2`) para eliminar los elementos repetidos de forma que la eliminación se haga sobre el mismo array de entrada, sin usar un segundo array auxiliar.

8. Haz un programa que obtenga la **mayor secuencia monótona creciente** de un array de enteros leído de la entrada estándar, guardándola en otro array que se mostrará en la salida estándar.

9. Implemente una función `mezclarUnico` de tipo `int` que reciba como entrada dos arrays ordenados de datos de tipo `double` y los mezcle en un tercer array. Tenga en cuenta que:

   - Los arrays de entrada están ordenados y sin valores repetidos.
   - El array de salida tendrá los elementos ordenados y sin repetidos.

- Puede asumir que el array de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el array de salida. Nota: Será menor o igual que la suma de los de entrada.

Casos de prueba

Los casos de prueba a considerar son:

```
// Caso 1
double array1[]={1, 3, 5, 7};
double array2[]={2, 4, 4.3, 9};

// Caso 2
double array3[]={1, 3, 5};
double array4[]={2, 3.8, 4.3, 6.4, 9.3};

// Caso 3
double array5[]={5, 6.3, 7.5, 8.3, 9.2};
double array6[]={1.0, 3.4, 6.3};
```

La salida que debería obtenerse es:

```
Caso de prueba 1:
1 3 5 7
2 4 4.3 9
```

```
1 2 3 4 4.3 5 7 9
```

```
Caso de prueba 2:
1 3 5
2 3.8 4.3 6.4 9.3
```

```
1 2 3 3.8 4.3 5 6.4 9.3
```

```
Caso de prueba 3:
5 6.3 7.5 8.3 9.2
1 3.4 6.3
```

```
1 3.4 5 6.3 7.5 8.3 9.2
```

---

10. Haz un programa que lea un texto desde **cin** y que, al finalizar la entrada de datos, muestre **cuántas veces aparece cada letra (a..z)**, teniendo en cuenta lo siguiente:

- la entrada de datos finaliza con el carácter #.
- no se diferenciarán mayúsculas y minúsculas.
- el texto se lee carácter a carácter en una función que guarda los caracteres en un array de **char** recibido como parámetro.
- usaremos una función que reciba como parámetro un array de caracteres y construya un array de enteros con la frecuencia para cada carácter.

11. Haz un programa que contenga una función que reciba un primer parámetro de tipo cadena de caracteres (estilo C) y que devuelva una **copia** de la misma en un segundo parámetro.

12. Haz un programa que contenga una función que compruebe si una cadena (estilo C) es o no un **palíndromo**, es decir, que se lee igual de izquierda a derecha que de derecha a izquierda sin tener en cuenta los espacios. Por ejemplo, **anilina** sería un palíndromo. La función para ver si la cadena es un palíndromo, se ayudará de otra función auxiliar que, dada una cadena, obtiene otra donde se han eliminado los espacios en blanco.

13. Haz un programa que contenga una función que reciba como entrada dos cadenas de caracteres estilo C (cad y subcad) y que compruebe si la segunda (subcad) está presente o no dentro de la primera (cad). Si la subcadena es encontrada, devolverá la posición donde se encuentra y, en caso contrario, devolverá -1.

14. Construir una función que **inserte una cadena de caracteres** dentro de otra cadena en una determinada posición.

15. Haz un programa que calcule, mediante una función, la **traza de una matriz** 2D (suma de los elementos de la diagonal principal).

16. Haz un programa que calcule la **traspuesta de una matriz** 2D. La inversión se hace por medio de una función.

17. Calcular el **producto de dos matrices** A y B, suponiendo que A tiene m filas y n columnas y B tiene n filas y p columnas.

2 Problemas complementarios

1. Supongamos que se eligen dos puntos de forma aleatoria (con distribución uniforme) en el cuadrado unitario $[0, 1]^2$. Escribid un programa que estime la longitud media esperada

del segmento que une dichos puntos. Hacedlo de forma que los resultados de cada prueba (cada selección de par de números) se almacenen en un array. La implementación debe ser lo más modular posible, incluyendo funciones auxiliares necesarias para todas las tareas que se consideren oportunas.

2. Escribid un programa que rellene dos arrays a y b con enteros, obtenidos de acuerdo a la siguiente recurrencia:

$$\begin{aligned} a_0 = b_0 &= 1 \\ a_n &= b_{n-1} \\ b_n &= a_{n-1} + 2b_{n-1} \end{aligned}$$

   El programa debe generar una tabla en la que cada fila tendrá la siguiente información:

$$k \;\; a_k \;\; b_k \;\; a_k/b_k$$

   Haz una conjetura sobre el valor de $\lim_{n->\infty} \frac{a_n}{b_n}$. El código resultante debe estar modularizado, incluyendo funciones auxiliares para todas las tareas que se consideren oportunas

3. Escribid una definición apropiada de arrays para cada uno de los siguientes problemas:

   (a) definición de una colección de 12 componentes enteros llamado **array**. Se asignarán los valores $1, 4, 7, 10, \ldots, 34$ a la colección.

   (b) definición de un array de cuatro caracteres llamado **letras**. Se asignarán los valores 'H', 'o', 'l', 'a' al mismo.

   (c) definición de un array de 6 elementos llamado **valor**. Se asignan los valores 1, $\frac{1}{2}$, $\ldots$, $\frac{1}{6}$ al mismo

4. Dado un array de números reales, realizar una función que determine el primer y segundo elemento de mayor valor.

5. Construir una función que permita ordenar de forma descendente los elementos de un array usando el algoritmo de inserción.

6. Se pide construir una función que tenga como entrada un array de caracteres (tipo **char**), y suprima todas las secuencias de espacios en blanco de longitud mayor de uno. Por ejemplo, si el array original es **(' ','a','h',' ',' ',' ','c')**, el array resultante debe ser **(' ','a','h',' ','c')**. Las modificaciones se harán sobre el mismo array de entrada y no se podrán usar arrays auxiliares. La implementación debe ser lo más eficiente posible.

7. Construir una función que dada una cadena de caracteres que contiene un número entero, devuelva su correspondiente valor numérico. Razonar sobre qué hacer si la cadena no contiene un número entero.

8. Construir una función que dada una cadena de caracteres **cad** y dos valores enteros **pos** y **tam** modifique **cad** eliminando los **tam** caracteres empezando en la posición **pos**. Razonar sobre todos los casos posibles que pueden suceder.

9. Construir una función que ordene alfabéticamente un array de cadenas de caracteres.

10. Realizar una función que lea un número natural y lo traduzca a morse. Se usa raya (**-**) para codificar la señal larga y punto (**.**) para la señal corta. Los números se codifican de la forma siguiente:

    0 —— **1** ·—- **2** ··— **3** ···– **4** ····- **5** ····· **6** -···· **7** —··· **8** —-·· **9** ——-·

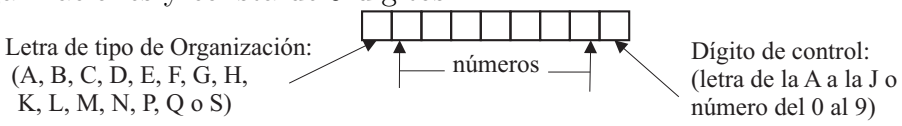
    De esta forma, cada número en morse se representará en C++ por una cadena de 5 caracteres compuesta de puntos y rayas, según la tabla anterior. Un ejemplo de lo que debe hacer el programa es:

```
Introduzca un número natural: 2005
 ..--- ----- ----- .....
Introduzca un número natural: 1992
 .---- ----. ----. ..---
```

11. Construir una función que reciba una cadena de caracteres y compruebe si contiene un número de identificación fiscal (NIF) o número de identidad de extranjero (NIE) válido. La función debe devolver el tipo de documento y si es o no válido. Decidir cómo se devuelve esta información.

    El NIF se forma con 8 dígitos seguidos de una letra (*dígito de control*). El procedimiento empleado para el cálculo del NIF consiste en hallar el resto de la división por 23 del número formado por los 8 dígitos. El resto resultante (comprendido entre 0 y 22) se corresponde con la letra en dicha posición (comenzando por 0) de la secuencia `TRWAGMYFPDXBNJZSQVHLCKE`. El NIE sigue el mismo procedimiento pero su sustituyendo el primer dígito del NIF por la letra X.

12. El CIF (Código de Identificación Fiscal) es un elemento de identificación administrativa para organizaciones y consta de 9 dígitos:



    El primer dígito es una letra que indica el tipo de la organización y puede ser una de los siguientes:

    A - Sociedades Anónimas. B - Sociedades de Responsabilidad Limitada. C - Sociedades Colectivas. D - Sociedades Comanditarias. E - Comunidades de Bienes. F - Sociedades Cooperativas. G - Asociaciones y otros tipos no definidos. H - Comunidades de propietarios en régimen de propiedad horizontal. N - Entidades no residentes. P - Corporaciones Locales. Q - Organismos Autónomos, estatales o no, y asimilados, y Congregaciones e Instituciones religiosas. S - Órganos de la Administración del Estado y Comunidades Autónomas.

    Los siete dígitos siguientes son números y el último es el *dígito de control* que puede ser una letra (en caso del CIF de Entidades no residentes (N), Corporaciones Locales (P), Organismos Autónomos (Q) ó Organismos de la administración (S)) o un número (en el caso de CIF de las restantes Sociedades).

Las operaciones para calcular el dígito de control se realizan sobre los siete dígitos centrales y son las siguientes:

(a) Sumar los dígitos de la posiciones pares. Suma = A

(b) Para cada uno de los dígitos de la posiciones impares, multiplicarlo por 2 y sumar los dígitos del resultado.
Ejemplo: (8 * 2 = 16 $\longrightarrow$ 1 + 6 = 7)
Acumular el resultado. Suma = B

(c) Sumar A + B = C

(d) Tomar sólo el dígito de las unidades de C y restárselo a 10. Esta resta nos da D.

(e) A partir de D ya se obtiene el dígito de control. Si ha de ser numérico es directamente D y si se trata de una letra se corresponde con la relación: A = 1, B = 2, C= 3, D = 4, E = 5, F = 6, G = 7, H = 8, I = 9, J = 10

Ejemplo para el CIF: Q1818009

(a) Utilizamos los siete dígitos centrales = 1818009

(b) Sumamos los dígitos pares: A = 8 + 8 + 0 = 16

(c) Posiciones impares:
1 * 2 = 2 $\rightarrow$ 2
1 * 2 = 2 $\rightarrow$ 2
0 * 2 = 0 $\rightarrow$ 0
9 * 2 = 18 $\rightarrow$1 + 8 = 9
Sumamos los resultados: B = 2 + 2 + 0 + 9 = 13

(d) Suma parcial: C = A + B = 16 + 13 = 29

(e) El dígito de las unidades de C es 9. Se lo restamos a 10 y nos da: D = 10 - 9 = 1

(f) Como el dígito de control ha de ser una letra es la "A".

Escribir una función que reciba una cadena de caracteres y devuelva si se trata de un CIF válido.

13. Como parte de un programa para imprimir el índice de un libro, se necesita una función que imprima cada línea. La función toma dos palabras almacenadas en cadenas de caracteres y la longitud de una línea **n** como un entero y escribe ambas palabras en una sola línea. Ambas palabras estarán separadas por puntos de tal forma que la longitud total de la línea sea **n** caracteres.

    Ejemplo:

    Primera palabra: **tortuga**
    Segunda palabra: **153**
    Longitud de la línea: 30
    tortuga.....153

14. Dadas dos cadenas de caracteres, construir una función en C++ que devuelva las veces que aparece la segunda cadena dentro de la primera. Reflexionar sobre qué debería devolver la función anterior, si las dos cadenas de entrada son **aaaaa** y **aaa**.

15. Responder apropiadamente a las siguientes cuestiones:

    (a) Definición de una matriz bidimensional 3×4 de enteros llamada **mat**. Asignar los siguientes valores a los elementos de la matriz.

    $$\begin{array}{cccc} 10 & 12 & 14 & 16 \\ 20 & 22 & 24 & 26 \\ 30 & 32 & 34 & 36 \end{array}$$

    (b) Definir una matriz bidimensional 3×4 de enteros llamada **mat**. Asignar los siguientes valores a los elementos de la matriz.

    $$\begin{array}{cccc} 10 & 12 & 14 & 16 \\ 20 & 22 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}$$

    (c) Definir una matriz tridimensional $10 \times 10 \times 10$ de enteros. Asignar en cada posición (i, j, k) la suma de los tres índices.

16. Realizar un programa modular que rellene una matriz bidimensional 7×5 de enteros con datos aleatorios e imprima por pantalla el número mayor y menor de la matriz y sus posiciones (fila y columna).

17. Construir una función para comprobar si dos matrices son iguales.

18. Construir una función para comprobar si una matriz es simétrica.

19. Haz un programa que sume dos matrices de enteros de tres dimensiones, usando una función auxiliar para realizar la suma.

20. En un congreso cuya duración es de 6 días, tienen lugar conferencias en 5 salas. Se desea saber:

    (a) El total de congresistas que asisten a cada una de las salas.

    (b) El total de congresistas asistentes cada día del congreso.

    (c) La media de asistencia a cada sala.

    (d) La media de asistencia diaria.

    (e) Imprimir para cada tabla la diferencia porcentual (positiva ó negativa) entre la asistencia diaria y la media de asistencia a cada sala.

    Como datos de entrada tendremos el número de asistentes para las diferentes salas, para cada uno de los días del congreso. Realizar una función que nos calcule estos datos.

21. Dado un array X de n elementos reales donde n es impar, diseñar una función para calcular la mediana de ese array. La mediana se define como el valor mayor que la mitad de los números y menor que la otra mitad.

    Por ejemplo, en el siguiente array:

    $$\begin{array}{cccccccccc} X[1] & X[2] & X[3] & X[4] & X[5] & X[6] & X[7] & X[8] & X[9] \\ 23 & 1 & 12 & 7 & 11 & 5 & -6 & -1 & 35 \end{array}$$

    que está compuesto de nueve elementos, la función debe encontrar que la solución es 7.

22. Representar un número binario en un array de datos de tipo lógico. Realizar las siguientes operaciones sobre números binarios expresados en esta forma:

    - Complemento a 2 de un número binario.
    - Suma y resta binaria.
    - Las operaciones lógicas AND y OR binarias (bit a bit).

23. Escribir un programa que calcule y muestre por pantalla los 20 primeros términos de la sucesión de Fibonacci de orden 4. El programa debe usar **obligatoriamente** una función que calcule y almacene en un array los k primeros términos de la sucesión de Fibonacci de orden n. Obviamente, k y n serán parámetros del módulo.

    La sucesión de Fibonacci de orden n es una secuencia de números en la que los dos primeros son el 0 y el 1. A partir del tercero, los elementos se calculan como la suma de los n anteriores, si ya hay n elementos disponibles, o la suma de todos los anteriores si hay menos de n elementos disponibles.

    Por ejemplo, la sucesión de Fibonacci de orden 4 sería la siguiente:

    $$0, 1, 1, 2, 4, 8, 15, 29, \ldots$$

24. Para obtener una lista de todos los números primos menores que un determinado número n, se puede utilizar la Criba de Eratóstenes. Ese método consiste en hacer una lista de todos los números desde 2 hasta $n - 1$. Tomamos el 2 y tachamos todos los múltiplos de 2. Luego tomamos el siguiente número que se encuentra después de 2 y que esté sin tachar, tachando de nuevo todos sus múltiplos. Repetimos este paso hasta que se acaben los números. Los números que quedaron sin tachar son los que no tienen divisores (salvo el 1 y él mismo) o sea los primos.
    Escribir una función que obtenga los números primos menores que un determinado número n utilizando el método anterior. La función tendrá como entrada un array de bool y el número n, y marcará en el array de bool con el valor *true* aquellas casillas cuya posición corresponda con un número primo (menor a n) y con el valor *false* las posiciones que no correspondan con números primos. La posiciones 0 y 1 las marcaremos con *true*.

    Por ejemplo, si $n = 11$ entonces el array contendrá los valores:

0	1	2	3	4	5	6	7	8	9	10
true	true	true	true	false	true	false	true	false	false	false

25. Realizar una función que acepte una matriz de enteros y devuelva el número de columnas *únicas* de la matriz, es decir, aquellas para las que **NO** existe otra columna en la matriz con los mismos valores.

    Por ejemplo, dada la matriz 4×7

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

la función deberá devolver 2, ya que las únicas columnas no repetidas son la tercera y la sexta.

26. Realizar una función que construya un array conteniendo todos los elementos menores a 100 del conjunto C definido a continuación:

    (a) El número 1 pertenece a C.

    (b) Si x pertenece a C, entonces $2x + 1$ y $3x + 1$ también pertenecen a C.

    (c) Ningún otro número está en C.

    C tendrá el siguiente aspecto: $\{1, 3, 4, 7, 9, 10, \cdots\}$. El array resultado debe aparecer ordenado en orden creciente.