Ejercicios sobre *vectores*

Todos los ejercicios se resuelven empleando vectores clásicos (*arrays*). No es necesario, ni conveniente, emplear funciones para su resolución ya que el objetivo es trabajar directamente con *arrays*. Por lo tanto, bastará con escribir el código en la función `main`. En cualquier caso, podría emplearse alguna función siempre que no recibiera como argumento un vector.

3. Vamos a ampliar la funcionalidad de la aplicación desarrollada como solución al ejercicio 25 de la *Relación de Problemas II* (sucursal con mayor número de ventas).

   El programa debe leer un número indeterminado de datos de ventas (código de sucursal, código de producto y número de unidades vendidas).

   Nota: Recuerde que la lectura de datos finaliza cuando se introduce -1 como código de sucursal.

   La empresa dispone de **100 sucursales**, aunque algunas no han vendido ningún producto. Los códigos de los productos son caracteres entre `'a'` y `'j'`. El programa sólo leerá datos de venta de las sucursales que hayan realizado operaciones de ventas por lo que no todas las sucursales aparecerán.

   No se conoce a priori el número de operaciones de venta, ni el número de sucursales que se van a procesar, ni el código de las sucursales que se van a procesar, aunque sí se sabe que los códigos de éstas son números entre 1 y 100.

   Después de leer los datos de ventas el programa mostrará la sucursal que más unidades ha vendido y cuántas unidades.

   A continuación, presentará un listado en el que aparecerán código de sucursal y número total de productos vendidos en la sucursal. En el listado aparecerán únicamente las sucursales que hayan vendido algún producto.

   Finalmente mostrará el número total de operaciones de venta, el número de sucursales que hayan vendido algún producto y el número total de unidades vendidas.

4. Escribir un programa que lea un número indeterminado de caracteres (la lectura termina cuando se introduce un `*`) y muestre el número total de apariciones cada letra.

   a) Para poder leer un espacio en blanco **no** puede emplear `cin >> caracter`, sino `caracter = cin.get()`. Cada vez que se ejecute `cin.get()` se lee un carácter (incluido el espacio en blanco) desde la entrada de datos por defecto.

   b) Para el cómputo se consideran iguales una letra y la mayúscula corespondiente.

   c) No se contabilizan dígitos, separadores u otros caracteres.

Puede utilizar el texto del Quijote, disponible en: http://decsai.ugr.es/~carlos/FP/Quijote.txt

(La moda es la letra e con un total de 217900 apariciones)

5. Escribir un programa que lea un número indeterminado de caracteres (la lectura termina cuando se introduce un `*`) y los guarde en un vector. Finalmente, nos dirá si el vector es un **palíndromo**, es decir, que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, `{'a','b','b','a'}` sería un palíndromo.

6. *(Examen Febrero 2016)* Dado un vector de caracteres que contiene un mensaje cifrado, se pide construir otro vector nuevo con el mensaje descifrado y mostrarlo.

   La forma de descifrado consiste en coger la primera y última letra de cada palabra. Las palabras están separadas por uno o más espacios en blanco o el final del vector. Si la última palabra no tiene espacios en blanco a su derecha, se coge sólo el primer carácter.

   Por ejemplo, si denotamos el inicio y final de la secuencia con un corchete, entonces:
   `[ Hidrógeno limpia ]` se descodificaría como `[Hola]`

   Reserve memoria para trabajar con un máximo de 1000 caracteres.

   Ejemplo de entrada: `[Hidrógeno limpia ]` – Salida correcta: `[Hola]`

   Ejemplo de entrada: `[Hidrógeno limpia]` – Salida correcta: `[Hol]`

   Ejemplo de entrada: `[Hidrógeno]` – Salida correcta: `[H]`

   Ejemplo de entrada: `[Hidrógeno ]` – Salida correcta: `[Ho]`

   Ejemplo de entrada: `[H]` – Salida correcta: `[H]`

   Ejemplo de entrada: `[H ]` – Salida correcta: `[H]`

7. Construya la función con cabecera:

   ```
   string IntToString (int n)
   ```

   para que convierta y devuelva el entero `n` a un `string`.

8. *(Examen Febrero 2015)* Se dispone de una serie de datos pluviométricos de varios años, en concreto del número de litros que han caído en un día en cada una de las ciudades del mundo. Se quiere calcular los k mayores índices pluviométricos, ordenados de mayor a menor.

   Se pide implementar el siguiente algoritmo: Construya un programa que vaya leyendo datos desde teclado hasta que se introduzca `-1`. A continuación lea el número k y aplique el siguiente algoritmo:

```
Ordenar el vector de mayor a menor
(se recomienda usar el algoritmo de ordenación por inserción)

Seleccionar los k primeros e introducirlos en topk
```

Finalmente, imprima los valores del vector `topk` en pantalla.

Por ejemplo, para la secuencia de datos `2.1 0.7 12.4 2.6 3.2` y $k = 2$ mostrará como resultado: `12.4 3.2`

9. Escribir un programa que lea un número indeterminado de números positivos (termina la lectura cuando se introduce un negativo) aunque nunca leerá más de 50. Conforme los va leyendo, los va almacenando en un vector, `datos`.

   A continuación elimina del vector los valores repetidos, dejando una sola copia. No se dejan huecos en el vector y todos los números quedan agrupados en las posiciones más bajas del vector.

   Implementar **tres** versiones del borrado:

   a) Usar un **vector auxiliar** `sin_repetidos` en el que almacenamos una única aparición de cada componente:

   ```
   Copiar la primera componente de "datos" en "sin_repetidos"
   Desde la segunda casilla de "datos", hasta la última:
      Si el valor de la casilla NO está en "sin_repetidos",
        añadirla al vector "sin_repetidos"
   Sustituir todas las componentes de "datos"
        por las de "sin_repetidos"
   ```

   b) El problema del algoritmo anterior es que usa otro vector, lo que podría suponer una carga importante de memoria si trabajásemos con vectores grandes. Por lo tanto, vamos a resolver el problema sin usar vectores auxiliares.

   Si una componente está repetida, **se borrará del vector**. Para borrar la componente de la casilla `p`:

   ```
   Desde la casilla "p" hasta la penúltima:
    Copiar en la casilla "p" el contenido de la "p"+1
   ```

   c) El anterior algoritmo nos obliga a desplazar muchas componentes. Implementar el algoritmo que usa dos *apuntadores*, `posicion_lectura` y `posicion_escritura`, que nos van indicando, en cada momento, la componente que se está leyendo y el sitio dónde tiene que escribirse.

   Por ejemplo, supongamos que en un determinado momento la variable `posicion_lectura` vale 6 y `posicion_escritura` 3. Si la componente en la posición 6 está repetida, simplemente avanzaremos `posicion_lectura`. Por el contrario, no estuviese repetida, la colocaremos en la posición 3 y avanzaremos una posición **ambas** variables.

10. *(Examen Septiembre 2012)* La ***criba de Eratóstenes*** (Cirene, 276 a. C. Alejandría, 194 a. C.) es un algoritmo que permite hallar todos los números primos menores que un número natural dado n.

    El procedimiento "manual" consiste en escribir todos los números naturales comprendidos entre 2 y n y *tachar* los números que *no* son primos de la siguiente manera: el primero (el 2) se declara primo y se tachan todos sus múltiplos; se busca el siguiente número entero que no ha sido tachado, se declara primo y se procede a tachar todos sus múltiplos, y así sucesivamente. El proceso de criba termina cuando el cuadrado del número entero es mayor o igual que el valor de n.

    Si `MAX_PRIMO` es el mayor número que se va a considerar, hacer un programa que lea el valor n ($1 \leq n \leq$`MAX_PRIMO`) y que calcule y muestre todos los primos menores o iguales que n.

    Recomendaciones:

    a) Para realizar la criba use el vector `es_primo` de datos `bool` de manera que si `es_primo[i]` es `true` entonces, i es primo.

    b) El resultado (los números primos calculados) se guardarán en el vector `primos`, de datos `int`, almacenados de manera consecutiva, sin huecos. Se establece un tamaño máximo para el vector de `MAX_DATOS` casillas.

    c) Tenga en cuenta las posibles situaciones de error, detéctelas y actúe correctamente.

11. En el ejercicio 30 de la *Relación de Problemas II* trabajamos sobre la codificación **RLE** (Run Length Encoding), que codifica una secuencia de datos formada por series de valores idénticos consecutivos como una secuencia de parejas (valor, número de veces que se repite).

    En ese ejercicio estábamos obligados a procesar los datos conforme se leían ya que no podíamos almacenarlos. Ahora que conocemos cómo guardar en un array una serie de datos es posible separar claramente las fases de lectura, procesamiento y presentación de resultados.

    Escriba un programa que haga, en este orden:

    a) Leer y almacenar en un vector `s` un número indeterminado (no más de 50) de **caracteres**.

    La secuencia se lee en una cadena de caracteres (use un dato `string`). Las cadenas vacías o de longitud mayor que 50 no se consideran.

    Un ejemplo de secuencia sería:

    `aaabddeeeeeddsssssa`

    Mostrar la secuencia original.

b) **Codificar** mediante el método RLE el vector `s`. Guardar el resultado de la codificación en otro vector (`s_codificada`).

Mostrar la secuencia codificada.

c) Usando el vector `s_codificada` **descodificar** la secuencia y almacenar el resultado en un tercer vector (`s_reconstruida`).

Mostrar la secuencia reconstruida.

d) Comparar las secuencias `s` y `s_reconstruida` e informar del resultado de la comparación.

12. *(Examen Noviembre 2016)* Escribir un programa que genere números aleatorios hasta acertar un número establecido por el usuario.

El programa evaluará la calidad del algoritmo empleado. Para ello, se realizará una serie indeterminada de experimentos (nunca mayor de 50) y en base al número de intentos totales se determinará la "calidad" del juego.

Los detalles del juego:

a) El programa pedirá dos números enteros positivos a y b. El juego considerará únicamente valores comprendidos entre a y b.

Si el número de valores comprendidos entre a y b (ambos incluidos) es menor que un valor preestablecido (por ejemplo, 10) no se hace nada más.

b) El programa pedirá al usuario los números que después intentará acertar.

El usuario introducirá una secuencia indeterminada de n números enteros (no más de 50) comprendidos entre a y b, y los almacena en un **vector**. Los números positivos fuera de rango no se consideran. Si se introduce un número negativo o se excede del número máximo permitido de números a procesar, se finaliza la lectura.

c) Cada experimento de acierto consiste en acertar uno de los números guardados en el vector:

I) El programa genera números aleatorios (entre a y b) hasta acertar.

II) Se cuenta el número de intentos realizados.

d) El valor de referencia usado para la evaluación es el número **total** de intentos, t.

Si n es el número de experimentos realizados y k es el número de valores comprendidos entre a y b (ambos incluidos), los valores a tener en cuenta para etiquetar el resultado del juego están comprendidos entre:

- n, el menor valor posible (supone una suerte extraordinaria), y
- $n \times k$, el "peor" valor posible en el caso de hacer recorridos exhaustivos y obtener los peores resultados posibles en cada uno de ellos.

Dividimos el intervalo $[n,\ n \times k]$ en cinco intervalos que comprende, cada uno de ellos, el $20\,\%$. Si t es el número total de intentos efectuados, el resultado se etiquetará como: a) **Bueno**, b) **Medio-bueno**, c) **Medio**, d) **Medio-malo** ó e) **Malo**, de acuerdo al tramo en el que se encuentre t en el intervalo indicado.

El programa deberá indicar la etiqueta que corresponde.

Tenga en cuenta que t podría ser mayor que $n \times k$, y en este caso el programa indicará que el resultado es **Muy malo**.

Por ejemplo, para $a = 21$ y $b = 120$ ($k = 100$), presentamos dos posibles resultados habiendo realizado $n = 10$ experimentos:

```
TOTAL EXPERIMENTOS =   10
TOTAL GENERADOS    =  879

Bueno       =    10.00 <-->   210.00
Medio-bueno =   210.00 <-->   421.00
Medio       =   421.00 <-->   621.00
Medio-malo  =   621.00 <-->   800.00
Malo        =   800.00 <-->  1000.00
Muy malo    =  1000.00 <-->  oo

El resultado global ha sido: MALO
```

```
TOTAL EXPERIMENTOS =   10
TOTAL GENERADOS    =  411

Bueno       =    10.00 <-->   210.00
Medio-bueno =   210.00 <-->   421.00
Medio       =   421.00 <-->   621.00
Medio-malo  =   621.00 <-->   800.00
Malo        =   800.00 <-->  1000.00
Muy malo    =  1000.00 <-->  oo

El resultado global ha sido: MEDIO-BUENO
```

13. En este ejercicio se trata de resolver el mismo problema que el planteado en el ejercicio 29 de la *Relación de Problemas II*.

    En este caso, la secuencia de temperaturas se guarda en un **vector** de datos `double`. El cálculo (la subsecuencia de números ordenada, de menor a mayor, de mayor longitud) lo realizará tomando los datos del vector.

Ejercicios sobre *matrices*

Todos los ejercicios se resuelven empleando matrices clásicas. No es necesario, ni conveniente, emplear funciones para su resolución ya que el objetivo es trabajar directamente con los vectores y las matrices. Por lo tanto, bastará con escribir el código en la función `main`. En cualquier caso, podría emplearse alguna función siempre que no recibiera como argumento una matriz.

14. En este problema se propone reescribir otra solución del problema de las sucursales (ejercicio 3 de esta misma relación de problemas). Queremos almacenar en memoria no sólo las ventas totales de cada sucursal (resuelto en el ejercicio indicado anteriormente), sino la información referente a los productos vendidos. En definitiva:

    - Total de unidades vendidas de cada producto.
    - Producto más vendido.
    - Cuántos tipos de producto han sido vendidos (número de tipos de producto con unidades vendidas mayores que cero).
    - Número total de unidades vendidas entre todas las sucursales.

    Para poder guardar en memoria la información requerida para los cálculos proponemos una matriz bidimensional `ventas` con tantas filas como número (máximo) de sucursales y columnas como número (máximo) de productos. La casilla (s, p) de esta matriz guardará en número total de unidades vendidas por la sucursal s del producto p (o el número de de unidades vendidas del producto p en la sucursal s).

    No se conoce a priori el número de operaciones de venta, ni el número de sucursales que se van a procesar, ni el código de éstas (algunas sucursales puede que no hayan realizado ventas). Se sabe que los códigos de sucursales son números entre 1 y 100.

    Tampoco se conoce a priori los productos que se han vendido, ni el código de éstos (algunos productos puede que no hayan vendido). Se sabe que los códigos de producto son caracteres entre `'a'` y `'j'`.

    Nota: Emplead datos `int` para los índices de las filas y `char` para las columnas, de manera que se accede a las ventas del producto `'b'` por la sucursal 3, por ejemplo, con la construcción: `ventas[3]['b']`.

    Nota: Aconsejamos que una vez leídos los datos, y actualizada la matriz `ventas`:

    a) Usar un vector, `ventas_sucursal`, con tantas casillas como filas tenga la matriz `ventas`. Guardará el número total de unidades vendidas por cada sucursal.

    b) Usar un vector, `ventas_producto`, con tantas casillas como columnas tenga `ventas`. Guardará el número total de unidades vendidas de cada producto.

15. Definir las constantes que especifican el número de filas y columnas de la matriz (su *capacidad*) y leer (filtrando adecuadamente) los valores de dos variables `filas` y `columnas` que contendrán el número de filas y columnas que se utilizarán.

    Leer los `filas` $\times$ `columnas` datos de la matriz, y realizar las siguientes tareas:

    a) Calcular la traspuesta de la matriz, guardando el resultado en otra matriz (siempre que sea posible).

    b) Leer los datos de otra matriz (incluidas sus dimensiones) que tenga la misma capacidad y multiplicar ambas matrices. Guardar el resultado en otra matriz (siempre que sea posible efectuar la multiplicación).

16. *(Examen Septiembre 2011)* Definir las constantes que especifican el número de filas y columnas de la matriz (su *capacidad*) y leer (filtrando adecuadamente) los valores de dos variables `filas` y `columnas`.

    Leer los `filas` $\times$ `columnas` datos de la matriz, y calcular la posición del elemento que sea el mayor de entre los mínimos de cada fila.

    Por ejemplo, dada la matriz M (3×4), el máximo entre 4, 2 y 1 (los mínimos de cada fila) es 4 y se encuentra en la posición (0,2).

9	7	4	5
2	18	2	12
7	9	1	5

17. Seguimos trabajando sobre la codificación RLE. En esta ocasión se va a codificar una serie indeterminada (no más de 20) de secuencias de caracteres de las características descritas en el problema 11 de esta misma relación de problemas.

    Cada secuencia se lee en una cadena de caracteres (use un dato `string`). Las cadenas vacías o de longitud mayor que 50 no se consideran. Para terminar de introducir secuencias bastará con escribir `FIN`.

    Por ejemplo, la siguiente entrada proporciona al programa dos secuencias:

    ```
    aaabddeeeeeddssssssa

    bbdttttrrrrrr
    aaaaaaaaaaaaaaaaaaaaaaasssssssssssssssssssssfffffffffffffffff
    FIN
    ```

    (Solo se consideran dos, porque una está *vacía* y otra tiene más de 50 caracteres).

    Escriba un programa que haga, en este orden:

    a) Leer y almacenar en la matriz `m` un número indeterminado (no más de 20) secuencias de (no más de 50) **caracteres** (cada secuencia se guarda en una fila).
    Mostrar las secuencias que se van a procesar.

b) **Codificar** mediante el método RLE.

Guardar el resultado de la codificación en otra matriz (`m_codificada`). Cada secuencia codificada se guarda en una fila.

Mostrar las secuencias codificadas.

c) Usando la matriz `m_codificada`, **descodificar** las secuencias y almacenar el resultado en una tercera matriz (`m_reconstruida`). Cada secuencia descodificada se guarda en una fila.

Mostrar las secuencias reconstruidas.

d) Comparar las secuencias guardadas en `m` y `m_reconstruida` e informar del resultado de la comparación.

18. "Dibujar" un círculo, dado un centro y un radio.

En realidad se trata de hacer algo más *simple*, para lo que tengamos herramientas, pero el resultado se asemejará a un dibujo hecho con trazo grueso: si mostramos el contenido de una **matriz de caracteres** (habiendo elegido estratégicamente *los caracteres* y *dónde colocarlos*) podemos obtener resultados muy interesantes. Es lo que se conoce como **ASCII art**. En la figura 17 mostramos unos ejemplos.

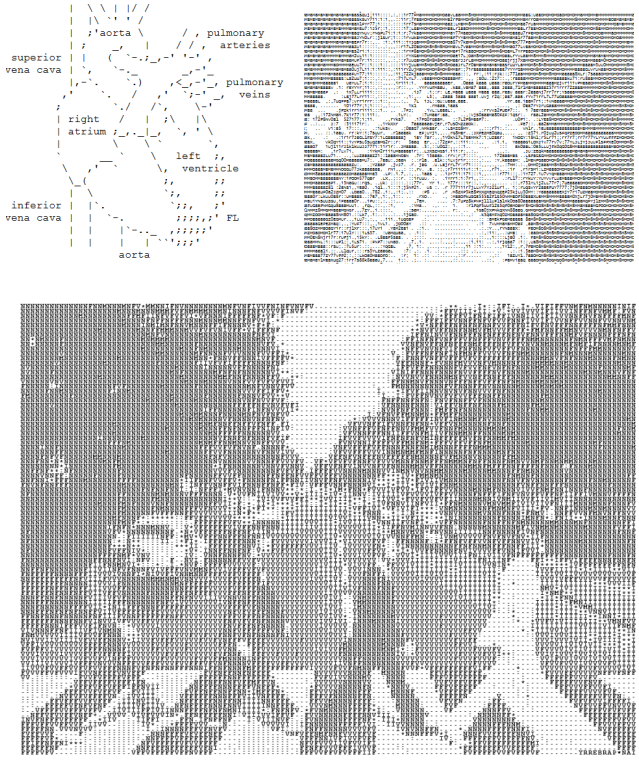

Figura 17: Ejemplos de ASCII art

El proceso de dibujo consiste en mostrar el contenido de la matriz recorriéndola por filas. Esto significa que la matriz es el tablero de dibujo, en la que la casilla de la fila 0 y columna 0 se corresponde con la esquina superior izquierda del dibujo, que cada "punto" del dibujo es una casilla de la matriz y que cada tipo de "trazo" es un carácter diferente.

Nuestro propósito es hacer un dibujo sencillo:

- binario, en el que sólo hay dos trazos o caracteres: espacio en blanco (no hay trazo) y $*$ (hay trazo).
- generado automáticamente, en el que cada "punto" es dibujado o no según el resultado de una función/test lógico.

Para simplificar suponemos que:

a) el tablero tiene `NUM_FILAS=51` filas y `NUM_COLS=51` columnas,

b) asociamos el tablero al primer cuadrante, esto es, el origen de coordenadas $(0, 0)$ estará asociado a la esquina inferior izquierda del tablero, la casilla `[50][0]`

El programa pedirá el valor del centro de la circunferencia y el radio, y dibujará los puntos que configuran el círculo.

Importante: Es posible que parte del círculo quede fuera del tablero. Debe dibujar lo que sea posible ser dibujado (los puntos que quedan dentro del tablero).

En la figura 18 mostramos dos ejemplos de ejecución.

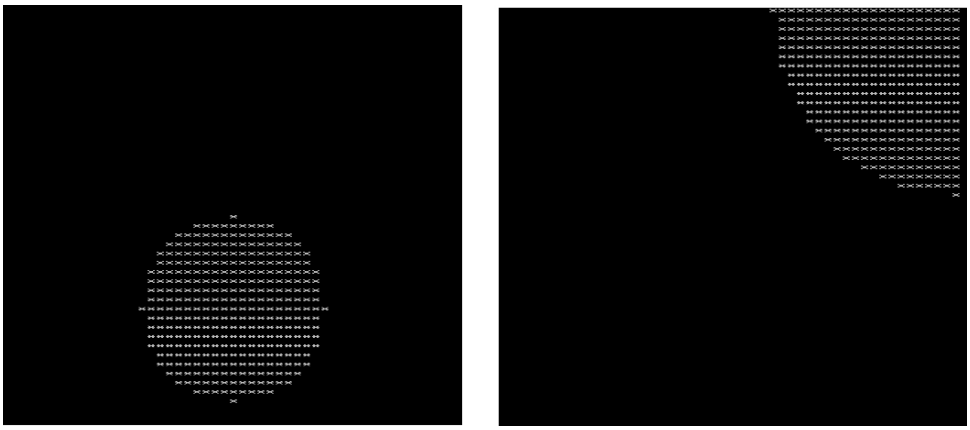

Figura 18: Izquierda: centro = (25, 10), radio = 10. Derecha: centro = (50,50), radio=20