RELACIÓN DE PROBLEMAS IV. Vectores

En los ejercicios que pida trabajar sobre la clase `SecuenciaCaracteres`, use la siguiente definición:

```
class SecuenciaCaracteres {
private:
    static const int TAMANIO = 50;
    char vector_privado[TAMANIO];
    int total_utilizados;
public:
    SecuenciaCaracteres()
      :total_utilizados(0){
    }
    int TotalUtilizados(){
        return total_utilizados;
    }
    int Capacidad(){
        return TAMANIO;
    }
    void Aniade(char  nuevo){
        if (total_utilizados < TAMANIO){
            vector_privado[total_utilizados] = nuevo;
            total_utilizados++;
        }
    }
    char Elemento(int indice){
        return vector_privado[indice];
    }
    void Elimina (int posicion){
      if (posicion >= 0 && posicion < total_utilizados){
         int tope = total_utilizados-1;

         for (int i = posicion ; i < tope ; i++)
            vector_privado[i] = vector_privado[i+1];

         total_utilizados--;
      }
   }
    string ToString(){
       string cadena;
```

```
        for (int i=0; i<total_utilizados; i++)
           cadena = cadena + vector_privado[i];

        return cadena;
    }
};
```

Importante:

- Para todos los ejercicios, se ha de diseñar una batería de pruebas.
- Recuerde lo visto en las transparencias del primer tema: para poder leer un espacio en blanco **no** puede emplear `cin >> caracter`, sino `caracter = cin.get()`. Cada vez que se ejecute `cin.get()` el compilador lee un carácter (incluido el espacio en blanco, el tabulador y el retorno de carro `'\n'`) desde la entrada de datos por defecto. En definitiva, el bucle de lectura de datos será del tipo:

  ```
  caracter = cin.get();

  while (caracter != TERMINADOR){
     ......
     caracter = cin.get();
  }
  ```

  Supondremos que la entrada de datos es desde un fichero de texto. De esta forma, cada ejecución de `cin.get()` lee directamente un carácter (incluidos los espacios en blanco, tabuladores y retornos de carro) y el programa pasa a la siguiente sentencia. Si fuese desde el teclado, habría que esperar a que el usuario introdujese el retorno de carro para que los datos pasasen al buffer y una vez ahí, se ejecutarían automáticamente todos los `cin.get()` (recordad lo visto al final del primer tema) En el caso de que, por ejemplo, quisiéramos parar la lectura cuando se hubiesen introducido más de un número tope de caracteres, no podríamos hacerlo con la lectura desde teclado ya que los datos no pasan al buffer hasta que no se pulse el retorno de carro.

1. Tenga en cuenta la observación al inicio de esta relación de problemas sobre la lectura de los caracteres. Para poder leer caracteres, incluyendo los espacios en blanco, hay que usar `caracter = cin.get()`, en vez de `cin >> caracter`.

   En este ejercicio trabajaremos con un vector directamente en el `main`, sin utilizar clases.

   Declare un vector de caracteres de tamaño 100. Lea las componentes considerando como terminador el carácter # (éste no forma parte de la secuencia) y que no se introduzcan más de 100 caracteres. Las componentes leídas ocuparán las primeras posiciones contiguas del vector. El resto de las posiciones se quedarán con el valor indeterminado (basura) que el compilador le asignase al principio. Para conocer cuántas componentes se están utilizando, utilice una variable `total_utilizados` (que, obviamente, deberá ser menor de 100 en todo momento)

   Implemente algoritmos para realizar las siguientes tareas:

   a) Comprobar 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, pero `{'a','c','b','a'}` no lo sería. Si la secuencia tiene un número impar de componentes, la que ocupa la posición central es descartada, por lo que `{'a','b','j','b','a'}` sería un palíndromo.

   b) Invertir el vector. Si éste contenía, por ejemplo, los caracteres `{'m','u','n','d','o'}`, después de llamar al método se quedará con `{'o','d','n','u','m'}`.

   c) Contar el número de mayúsculas que contiene.

   Finalidad: Recorrer las componentes de un vector. *Dificultad Baja.*

2. Construya la función con cabecera:

   ```
   string Digitos (int n)
   ```

   para que extraiga en un `string` los dígitos del número `n` tal y como se indica en el ejercicio 27 de la relación de problemas II.

   Finalidad: Trabajar con la clase `string`. *Dificultad Baja.*

3. Recupere la solución del ejercicio 28 (ventas de empresa) de la relación de problemas III. Resuelva el problema pedido (calcular la sucursal con mayor número de ventas) pero ahora considere que no conoce a priori el número de sucursales que hay, aunque sabe que los códigos de éstas siempre son números entre 1 y 100 y que en total no hay más de 100 sucursales. Por lo tanto, tendrá que añadir como dato miembro de la clase, un vector con un tamaño máximo de 100. Cree un programa principal de prueba.

   Finalidad: Trabajar con vectores como datos miembro de una clase. *Dificultad Baja.*

4. Añada los métodos `EsPalindromo`, `Invierte` y `NumeroMayusculas` a la clase `SecuenciaCaracteres` que implementen las tareas descritas en el ejercicio 1 de esta relación de problemas.

   Incluya un programa principal de prueba similar al del ejercicio 1.

   Finalidad: Diseñar las cabeceras de los métodos que acceden a las componentes del vector. Dificultad Baja.

5. Añada el método `IntercambiaComponentes` para intercambiar dos componentes de la secuencia. Por ejemplo, si la secuencia contiene `{'h','o','l','a'}`, después de intercambiar las componentes 1 y 3, se quedaría con `{'h','a','l','o'}`.

   ¿Qué debería hacer este método si los índices no son correctos?

   Modifique la implementación del método `Invierte` del ejercicio 4, para que lo haga llamando a `IntercambiaComponentes`.

   Imprima las componentes de la secuencia desde el `main`, antes y después de llamar a dicho método. Para ello, use el método `ToString()` de la clase `SecuenciaCaracteres`.

   Finalidad: Ejemplo de método que modifica el estado del objeto. Gestión de precondiciones. Dificultad Baja.

6. Añada el método `EliminaMayusculas` para eliminar todas las mayúsculas. Por ejemplo, después de aplicar dicho método al vector `{'S','o','Y',' ','y','O'}`, éste debe quedarse con `{'o',' ','y'}`.

   Un primer algoritmo para resolver este problema sería el siguiente (en ejercicios posteriores se verán métodos más eficientes):

   ```
   Recorrer todas las componentes de la secuencia
      Si la componente es una mayúscula, borrarla
   ```

   Queremos implementarlo llamando al método `Elimina` (que borra un único carácter):

   ```
   class SecuenciaCaracteres{
      .........
      void EliminaMayusculasError(){
         for (int i=0; i<total_utilizados; i++)
            if (isupper(vector_privado[i]))
               Elimina(i);
      }
   };
   ```

   El anterior código tiene un fallo. ¿Cuál? Pruébelo con cualquier secuencia que tenga dos mayúsculas consecutivas, proponer una solución e implementarla.

   Finalidad: Recorrido sencillo de un vector con dos bucles anidados. Dificultad Baja.

7. El algoritmo del ejercicio 6 es muy ineficiente ya que requiere trasladar muchas veces muchas posiciones (usa dos bucles anidados).

   Para resolver eficientemente este problema se propone utilizar dos variables, `posicion_lectura` y `posicion_escritura` que nos vayan 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 es una mayúscula, simplemente avanzaremos `posicion_lectura`. Por el contrario, si fuese una minúscula, la colocaremos en la posición 3 y avanzaremos una posición ambas variables.

   Implemente este algoritmo.

   Finalidad: Modificar un vector a través de dos **apuntadores**. *Dificultad Media.*

8. Añada un método `EliminaRepetidos` que quite los elementos repetidos, de forma que cada componente sólo aparezca una única vez. Se mantendrá la primera aparición, de izquierda a derecha. Por ejemplo, si la secuencia contiene
   `{'b','a','a','h','a','a','a','a','c','a','a','a','g'}`
   después de quitar los repetidos, se quedaría como sigue:
   `{'b','a','h','c','g'}`

   Implemente los siguientes algoritmos para resolver ester problema:

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

   ```
   Recorrer todas las componentes de la secuencia original
      Si la componente NO está en "sin_repetidos",
        añadirla (al vector "sin_repetidos")
   Asignar las componentes de "sin_repetidos" a la secuencia
   ```

   b) El problema del algoritmo anterior es que usa un vector local, 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 locales.

   Si una componente está repetida, **se borrará de la secuencia**. Para borrar una componente, llamamos al método `Elimina`.

   c) El anterior algoritmo nos obliga a desplazar muchas componentes cada vez que encontremos una repetida. Proponga una alternativa (sin usar vectores locales) para que el número de desplazamientos sea el menor posible e impleméntela.

   Consejo: Use la misma técnica que se indicó en el ejercicio 7 de eliminar las mayúsculas.

   Finalidad: Usar un vector local. Modificar un vector a través de dos apuntadores. *Dificultad Media.*

9. Añada un método `EliminaExcesoBlancos` para eliminar el exceso de caracteres en blanco, es decir, que sustituya todas las secuencias de espacios en blanco por un sólo espacio. Por ejemplo, si la secuencia original es `(' ','a','h',' ',' ',' ','c')`, que contiene una secuencia de tres espacios consecutivos, la secuencia resultante debe ser `(' ','a','h',' ','c')`.

   Nota: Debe hacerse lo más eficiente posible.

   Finalidad: Recorrido de las componentes de un vector, en el que hay que recordar lo que ha pasado en la iteración anterior. Dificultad Media.

10. En este ejercicio no hay que definir ninguna clase. Todas las operaciones se realizan directamente en el `main`.

    Construya un programa que vaya leyendo caracteres hasta que se encuentre un punto `'.'` y cuente el número de veces que aparece cada una de las letras mayúsculas. Imprimir el resultado.

    Una posibilidad sería declarar un vector `contador_mayusculas` con tantas componentes como letras mayúsculas hay (`'Z'-'A'+1`) y conforme se va leyendo cada carácter, ejecutar lo siguiente:

    ```
    cin >> letra;

    if (letra == 'A')
       contador_mayusculas[0] = contador_mayusculas[0] + 1;
    else if (letra == 'B')
       contador_mayusculas[1] = contador_mayusculas[1] + 1;
    else if (letra == 'C')
       contador_mayusculas[2] = contador_mayusculas[2] + 1;
    else ....
    ```

    Sin embargo, este código es muy redundante. Como solución se propone calcular de forma directa el índice entero que le corresponde a cada mayúscula, de forma que todos los `if-else` anteriores los podamos resumir en una **única** sentencia del tipo:

    ```
    contador_mayusculas[indice] = contador_mayusculas[indice] + 1;
    ```

    Hacedlo, declarando el vector directamente dentro del `main`.

    Finalidad: Acceder a las componentes de un vector con unos índices que representen algo. Dificultad Baja.

11. Sobre el ejercicio 10, construya una clase específica `ContadorMayusculas` que implemente los métodos necesarios para llevar el contador de las mayúsculas. Lo que se pretende es que la clase proporcione los métodos siguientes:

```
void IncrementaConteo (char mayuscula)
int  CuantasHay (char mayuscula)
```

Modifique el programa principal para que cree un objeto de esta clase y llame a sus métodos para realizar los conteos de las mayúsculas. Finalmente, hay que imprimir en pantalla cuántas veces aparece cada mayúscula.

Dificultad Media.

12. Construya una clase `CaminoComeCocos` para representar el camino seguido por el usuario en el juego del ComeCocos (Pac-Man). Internamente debe usar un vector de `char` como dato miembro privado. Tendrá métodos para subir, bajar, ir a la izquierda e ir a la derecha. Dichos métodos únicamente añadirán el carácter correspondiente `'s', 'b', 'i', 'd'` al vector privado.

    Añada a la clase un método `PosicionMovimientosConsecutivos` que calcule la posición donde se encuentre la primera secuencia de al menos n movimientos consecutivos iguales a uno dado (que pasaremos como parámetro al método).

    Por ejemplo, en el camino de abajo, si $n = 3$ y el movimiento buscado es `'s'`, entonces dicha posición es la 6.
    `{'b','b','i','s','s','b','s','s','s','s','i','i','d'}`

    Cree un programa principal que lea desde un fichero los caracteres que representan las posiciones hasta llegar a un punto (`'.'`), lea un carácter c y un entero n e imprima en pantalla la posición de inicio de los n movimientos iguales a c.

    Dificultad Baja.

13. Cread una clase `Permutacion` para representar una permutación de enteros. Para almacenar los valores enteros usaremos como dato miembro privado un vector clásico de enteros. La clase debe proporcionar, al menos, los siguientes métodos:

    - `Aniade` para añadir un número a la permutación.
    - `EsCorrecta` para indicar si los valores forman una permutación correcta, es decir, que contiene todos los enteros sin repetir entre el mínimo y el máximo de dichos valores. Por ejemplo, `(2,3,6,5,4)` es una permutación correcta pero no lo es `(7,7,6,5)` (tiene el 7 como valor repetido) ni tampoco `(7,6,4)` (le falta el 5).
    - `NumLecturas` para saber el número de lecturas de la permutación. Una permutación de un conjunto de enteros tiene k lecturas, si para leer sus elementos en orden creciente (de izquierda a derecha) tenemos que recorrer la permutación k veces. Por ejemplo, la siguiente permutación del conjunto $\{0, \ldots, 8\}$:

      4 0 8 1 2 5 3 6 7

      necesita 3 lecturas. En la primera obtendríamos 0, 1, 2 y 3. En la segunda 4, 5, 6 y 7 y finalmente, en la tercera, 8.

Cread un programa principal que lea desde un fichero los valores de la permutación e imprima el número de lecturas de dicha permutación.

Dificultad Media.

14. 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:

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

    Definid una clase llamada `Fibonacci`. Para almacenar los enteros, se usará un vector de enteros. Al constructor se le pasará como parámetro el valor de n. Definid los siguientes métodos:

    - `int GetBase()` para obtener el valor de n.
    - `void CalculaPrimeros(int tope)` para que calcule los `tope` primeros elementos de la sucesión.
    - `int TotalCalculados()` que devuelva cuántos elementos hay actualmente almacenados (el valor `tope` del método anterior)
    - `int k_esimo(int k)` para que devuelva el elemento `k`-ésimo de la sucesión.

    Escribid un programa que lea los valores de dos enteros, n y k y calcule, almacene y muestre por pantalla los k primeros términos de la sucesión de Fibonacci de orden n:

    ```
    ......
    Fibonacci fibonacci(n);

    fibonacci.CalculaPrimeros(k);
    tope = fibonacci.TotalCalculados();   // tope = k

    for (int i=0; i<tope; i++)
       cout << fibonacci.k_esimo(i) << " ";
    ```

    Dificultad Media.

15. *(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 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 para cuando el cuadrado del número entero es mayor o igual que el valor de n.

El programa debe definir una clase llamada `Eratostenes` que tendrá tres métodos:

- `void CalculaHasta(int n)` para que calcule los primos menores que `n`.
  Este es el método que implementa el algoritmo de Eratóstenes. Cuando se ejecute el método, se almacenarán en un vector interno del objeto todos los primos menores que `n`. Debe implementarse tal y como se ha indicado anteriormente, por lo que tendrá que decidir, por ejemplo, cómo gestiona el *tachado* de los números.
- `int TotalCalculados()` que devuelva cuántos primos hay actualmente almacenados.
- `int k_esimo(int k)` para que devuelva el `k`-ésimo primo.

El programa principal quedaría de la forma:

```
Eratostenes primos;
int n = 100; int num_primos;

primos.CalculaHasta(n);
num_primos = primos.TotalCalculados();

for (int i=0; i<num_primos; i++)
   cout << primos.k_esimo(i) << " ";
```

Dificultad Media.

16. Se van a gestionar las calificaciones de una clase formada por un número indeterminado de alumnos, aunque no superior a cien. Se pretende calcular la nota media final de cada alumno en base a **cuatro** calificaciones parciales con diferente peso.

    El programa leerá, en primer lugar, los pesos que se asignan a las calificaciones parciales (se esperan expresados en tantos por cien). Comprobad que las asignaciones son correctas y en el caso de que no lo fueran, abortad la ejecución del programa.

    A continuación leerá para cada alumno: *apellidos y nombre* (todo junto, separando apellidos y nombre por una coma, leedlos en un dato de tipo `string`) y las cuatro *calificaciones* (números reales entre 0.0 y 10.0 separados por espacios en blanco u otros separadores). La lectura finaliza cuando se introduce el caracter `*` en la lectura de los apellidos y nombre de un alumno.

    Una vez almacenados todos los datos leidos se mostrará un listado de: *apellidos y nombre* y *nota media* para cada alumno. El listado estará ordenado según la nota media final de cada alumno.

Reflexión: Piense cómo podría modificar el programa para que pueda considerar un número indeterminado de calificaciones (máximo 10).

a) El número de calificaciones será el primer dato que se lea, seguido de los pesos asignados a cada calificación.

b) A continuación del nombre de cada alumno aparecerán tantas calificaciones como indica el primer dato (número de pesos = número de calificaciones por alumno).

Recomendaciones:

a) Leer los datos usando la redirección de la entrada. Usad para ello un fichero de texto como el disponible en la página de la asignatura.

b) Es muy importante, por simplificar el problema aunque sin restarle generalidad, que los apellidos y nombres de cada alumno ocupen una sóla línea, sin más datos. Las calificaciones, no obstante, podrían estar separados en varias líneas y por un número indeterminado de separadores.

Dificultad Media.

17. *(Examen Febrero 2013)* Se está diseñando un sistema web que recolecta datos personales de un usuario y, en un momento dado, debe sugerirle un nombre de usuario (login). Dicho login estará basado en el nombre y los apellidos; en concreto estará formado por los N primeros caracteres de cada nombre y apellido (en minúsculas, unidos y sin espacios en blanco). Por ejemplo, si el nombre es `"Antonio Francisco Molina Ortega"` y N=2, el nombre de usuario sugerido será `"anfrmoor"`.

    Debe tener en cuenta que el número de palabras que forman el nombre y los apellidos puede ser cualquiera. Además, si N es mayor que alguna de las palabras que aparecen en el nombre, se incluirá la palabra completa. Por ejemplo, si el nombre es `"Ana CAMPOS de la Blanca"` y N=4, entonces la sugerencia será `"anacampdelablan"` (observe que se pueden utilizar varios espacios en blanco para separar palabras).

    Implemente la clase `Login` que tendrá como único dato miembro el tamaño `N`. Hay que definir el método `Codifica` que recibirá una cadena de caracteres (tipo `string`) formada por el nombre y apellidos (separados por uno o más espacios en blanco) y devuelva otra cadena con la sugerencia de login.

```
class Login{
private:
   int num_caracteres_a_coger;
public:
   Login (int numero_caracteres_a_coger)
```

```
        :num_caracteres_a_coger(numero_caracteres_a_coger)
    { }
    string Codifica(string nombre_completo){
        ......
    }
};
```

Los únicos métodos que necesita usar de la clase `string` son `size` y `push_back`. Para probar el programa o bien lea una cadena de caracteres con `getline(cin, cadena);` o bien use directamente literales string en el `main`.

Dificultad Media.

18. *(Examen Septiembre Doble Grado 2013)* Defina una clase `Frase` para almacenar un conjunto de caracteres (similar a la clase `SecuenciaCaracteres`). Defina un método para localizar la k-ésima palabra.

    Una palabra es toda secuencia de caracteres delimitada por espacios en blanco a izquierda y derecha. La primera palabra no tiene por qué tener espacios a su izquierda y la última no tiene por qué tener espacios a su derecha. Puede haber varios caracteres en blanco consecutivos.

    Si k es mayor que el número de palabras, se considera que no existe tal palabra.

    Por ejemplo, si la frase es `{' ',' ','h','i',' ',' ','b','i',' '}`. Si $k = 1$, la posición es 2. Si $k = 2$ la posición es 6. Si $k = 3$ la posición es -1.

    Si la frase fuese `{'h','i',' ','b','i',' '}`, entonces si $k = 1$, la posición es 0. Si $k = 2$ la posición es 3. Si $k = 3$ la posición es -1.

19. Sobre el ejercicio 18, añadid los siguientes métodos:

    - `void EliminaBlancosIniciales()` para borrar todos los blancos iniciales.
    - `void EliminaBlancosFinales()` para borrar todos los blancos finales.
    - `int NumeroPalabras()` que indique cuántas palabras hay en la frase.
    - `void BorraPalabra(int k_esima)` para que borre la palabra `k`-ésima.
    - `void MoverPalabraFinal(int k_esima)` para desplazar la palabra `k`-ésima al final de la frase.

    Dificultad Media.

20. *(Examen Septiembre Doble Grado 2013)* Defina la clase `SecuenciaEnteros` análoga a `SecuenciaCaracteres`. Defina lo que sea necesario para calcular el número de secuencias ascendentes del vector. Por ejemplo, el vector `{2,4,1,1,7,2,1}` tiene 4 secuencias que son `{2,4}, {1,1,7}, {2}, {1}`.

    Dificultad Media.

21. Implementad la **Búsqueda por Interpolación** en la clase `SecuenciaCaracteres`. El método busca un valor `buscado` entre las posiciones `izda` y `dcha` y recuerda a la *búsqueda binaria* porque requiere que el vector en el que se va a realizar la búsqueda esté ordenado y en cada consulta sin éxito se descarta una parte del vector para la siguiente búsqueda.

    La diferencia fundamental con la búsqueda binaria es la manera en que se calcula el elemento del vector que sirve de referencia en cada consulta (que ocupa la posición `pos`). Ya no es el que ocupa la posición central del subvector en el que se efectúa la búsqueda (el delimitado únicamente por `izda` y `dcha`), sino que depende también del contenido de esas casillas, de manera que `pos` será más cercana a `dcha` si `buscado` es más cercano a `v[dcha]` y más cercana a `izda` si `buscado` es más cercano a `v[izda]`. En definitiva, se cumple la relación:

$$\frac{\texttt{pos} - \texttt{izda}}{\texttt{dcha} - \texttt{izda}} = \frac{\texttt{buscado} - \texttt{v[izda]}}{\texttt{v[dcha]} - \texttt{v[izda]}}$$

22. Escriba un programa que rellene una matriz de dimension `MAX_FIL x MAX_COL` con números pares, lea del usuario una posición `(i,j)` y muestre por pantalla el valor de dicha posición. Nótese que es necesario controlar la posición introducida por el usuario.

    Finalidad: Manejar matrices. Dificultad Baja.

23. En este ejercicio, no hay que construir ninguna clase ni función. Es un ejercicio sobre recorridos de una matriz.

    Leed desde teclado dos variables `util_filas` y `util_columnas` y leed los datos de una matriz de enteros de tamaño `util_filas x util_columnas`. Sobre dicha matriz, se pide lo siguiente:

    a) Calcular la traspuesta de la matriz, almacenando el resultado en otra matriz.

    b) *(Examen Septiembre 2011)* La posición de aquel elemento que sea el mayor de entre los mínimos de cada fila. Por ejemplo, dada la matriz M (3×4),

$$\begin{matrix} 9 & 7 & 4 & 5 \\ 2 & 18 & 2 & 12 \\ 7 & 9 & 1 & 5 \end{matrix}$$

    el máximo entre 4, 2 y 1 (los mínimos de cada fila) es 4 y se encuentra en la posición $(0, 2)$.

    c) Ver si existe un valor *MaxiMin*, es decir, que sea a la vez, máximo de su fila y mínimo de su columna.

    d) Leer los datos de otra matriz y multiplicar ambas matrices (las dimensiones de la segunda matriz han de ser compatibles con las de la primera para poder hacer la multiplicación)

24. En este ejercicio, no hay que construir ninguna clase ni función. Es un ejercicio sobre recorridos de una matriz.

    Para ahorrar espacio en el almacenamiento de matrices cuadradas simétricas de tamaño $k \times k$ se puede usar un vector con los valores de la diagonal principal y los que están por debajo de ella. Por ejemplo, para una matriz $M = \{m_{ij}\}$ el vector correspondiente sería:

    $$\{m_{11}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33}, m_{41}, \cdots, m_{kk}\}$$

    Declarar una matriz clásica `double matriz[50][50]` en el `main`, asignarle valores de forma que sea cuadrada simétrica y construid el vector pedido. Haced lo mismo pero a la inversa, es decir, construir la matriz a partir del vector.

    Dificultad Media.

25. Escribir un programa que permita a dos jugadores jugar al tres-en-raya. El programa preguntará por los movimientos alternativamente al jugador `X` y al jugador `O`. El programa mostrará las posiciones del juego como sigue:

    $$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

    Los jugadores introducen sus movimientos insertando los números de posición que desean marcar. Después de cada movimiento, el programa mostrará el tablero cambiado. Un tablero de ejemplo se muestra a continuación.

    $$\begin{array}{ccc} \texttt{X} & \texttt{X} & \texttt{O} \\ 4 & 5 & 6 \\ \texttt{O} & 8 & 9 \end{array}$$

    El programa detectará al final de la partida si hay o no empate y en caso contrario, qué jugador ha ganado. Además, pedirá empezar una nueva partida y reiniciar el proceso.

    Finalidad: Practicar con el uso de matrices sencillas en una aplicación. *Dificultad Media.*

26. Escribir un programa para asignar asientos de pasajeros en un avión. Asumimos un avión pequeño con la numeración de asientos como sigue:

    $$\begin{array}{c|cc|cc|} 1 & A & B & C & D \\ 2 & A & B & C & D \\ 3 & A & B & C & D \\ 4 & A & B & C & D \\ 5 & A & B & C & D \\ 6 & A & B & C & D \\ 7 & A & B & C & D \\ \end{array}$$

El programa mostrará con una `X` el asiento que está ya asignado. Por ejemplo, después de asignar los asientos 1A, 2B, y 4C, lo que se mostrará en pantalla tendrá un aspecto como este:

1	X	*B*	*C*	*D*
2	*A*	X	*C*	*D*
3	*A*	*B*	*C*	*D*
4	*A*	*B*	X	*D*
5	*A*	*B*	*C*	*D*
6	*A*	*B*	*C*	*D*
7	*A*	*B*	*C*	*D*

Después de mostrar los asientos disponibles, el programa pregunta por el asiento deseado, el usuario teclea un asiento y el programa actualiza la asignación mostrando el esquema anterior. Primero se pide el número de fila y después la letra de asiento. Esto continua hasta que todos los asientos se asignen o hasta que el usuario indique que no quiere asignar más asientos (introduciendo el valor -1 en el número de la fila). Si el usuario introduce un asiento ya asignado, el programa mostrará un mensaje indicando que el asiento está ocupado y volverá a solicitarlo.

Finalidad: Practicar con el uso de matrices sencillas en una aplicación. Dificultad Baja.