RELACIÓN DE PROBLEMAS IV. Vectores

Los ejercicios básicos tienen como objetivo ampliar la clase `SecuenciaCaracteres`

```
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];
    }
};
```

Importante:

- Para todos los ejercicios, se ha de diseñar una batería de pruebas.
- 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) desde la entrada de datos por defecto.

Problemas Básicos

1. Añadid un método `EsPalindromo` a la clase `SecuenciaCaracteres` que nos diga si la secuencia 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. Incluya un progama principal que lea una serie de caracteres hasta llegar al terminador `'#'` y diga si es un palíndromo. Tenga en cuenta la observación de la página anterior sobre la lectura de los caracteres con `cin.get()`.

   Finalidad: Método que accede a las componentes del vector. Dificultad Baja.

2. Añadid un método con la cabecera:

   ```
   void Modifica (int indice_componente, char nuevo)
   ```

   para que sustituya la componente con índice `indice_componente` por el valor `nuevo`.

   Este método está pensado para modificar una componente ya existente, pero no para añadir componentes nuevas. Por tanto, en el caso de que la componente no esté dentro del rango correcto, el método no modificará nada.

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

3. Añadid 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?

   Añadid otro método llamado `Invierte` para invertir la secuencia. Si contenía, por ejemplo, los caracteres `{'m','u','n','d','o'}`, después de llamar al método se quedará con `{'o','d','n','u','m'}`. Implementad el método `Invierte` llamando a `IntercambiaComponentes`.

   Imprimid las componentes de la secuencia desde el `main`, antes y después de llamar a dicho método. Observad que se repite el mismo tipo de código cuando se imprimen las componentes de la secuencia. Ya lo arreglaremos en el tema siguiente.

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

4. Añadid el método `Elimina` para eliminar el carácter que se encuentre en una determinada posición. Si la secuencia contenía, por ejemplo, los caracteres

`{'h','o','l','a'}`, después de eliminar la componente con índice 2 (la tercera) se quedará con `{'h','o','a'}`. Para borrar una componente, se *desplazan* una posición a la izquierda todas las componentes que hay a su derecha.

¿Qué debería hacer el método si el índice no es correcto?

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

5. Añadid 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 se ha definido en el ejercicio 4 de esta relación de problemas:

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

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

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

6. El algoritmo del ejercicio 5 es muy ineficiente ya que requiere trasladar muchas veces muchas posiciones (usa dos bucles anidados). Proponer un algoritmo para resolver eficientemente este problema e implementarlo.

   Consejo: Una forma de hacerlo es 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. De esta forma, sólo necesitamos un bucle.

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

7. Añadid 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'}`

   Implementad los siguientes algoritmos para resolver ester problema:

   a) Usad 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" al
   vector original
   ```

   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, podríamos desplazar una posición a la izquierda, todas las componentes que estén a su derecha. Implementad el siguiente algoritmo:

   ```
   Recorrer todas las componentes de la secuencia original
      Si la componente se encuentra en alguna posición
      anterior, la eliminamos desplazando hacia la
      izquierda todas las componentes que hay a su derecha.
   ```

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

   Consejo: Usad la misma técnica que se indicó en el ejercicio 6 de eliminar las mayúsculas.

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

8. Añadid 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.

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

   Realizad un programa que vaya leyendo caracteres hasta que se encuentre un punto `'.'`. Contad el número de veces que aparece cada una de las letras mayúsculas e imprimirlo en pantalla.

   Una posibilidad sería declarar un vector `contador_mayusculas` y conforme se va leyendo cada carácter, ejecutar un conjunto de sentencias del tipo:

   ```
   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 la secuencia directamente dentro del `main`.

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

10. Sobre el ejercicio 9, cread 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)
    ```

    Modificad 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.

11. Construid 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.

Añadid 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'}`

Cread 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.

12. 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.

13. 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)

Problemas de Ampliación

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 Elemento(int indice)` para que devuelva el elemento `indice`-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.Elemento(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 Elemento(int indice)` para que devuelva el `indice`-ésimo primo.

El programa principal quedaría de la forma:

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

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

for (int i=0; i<tope; i++)
   cout << primos.Elemento(i) << " ";
```

Dificultad Media.

16. *(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){
      ......
   }
};
```

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.

17. *(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.

18. Sobre el ejercicio 17, 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.

19. *(Examen Septiembre Doble Grado 2013)* Defina la clase `SecuenciaEnteros` análoga a `SecuenciaCaracteres`. Defina lo que sea necesario para calcular el número de secuencias ascendientes 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.

20. 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:

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

21. *(Examen Septiembre 2014)* Existe un método para la clase `string` de C++, denominado `replace`, que cambia `n` caracteres de una cadena `cad1`, empezando en una determinada posición `pos`, por los caracteres presentes en una segunda cadena `cad2`. La llamada al método es `cad1.replace(pos, n, cad2)`. Ejemplos del funcionamiento de `replace` son:

```
string cad1="Fundamental Programación";
cad1.replace(9,2,"os de la");   // "al" -> "os de la"
   // Ahora cad1 tiene "Fundamentos de la Programación"
cad1.replace(12,5,"en");        // "de la" -> "en"
   // Ahora cad1 tiene "Fundamentos en Programación"
```

    Sobre la clase `SecuenciaCaracteres` construid un módulo llamado `Reemplaza` con la misma funcionalidad que `replace`. No se puede utilizar la clase `string`.

22. 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.