RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

1. Recuperad las implementaciones de las clases `Punto2D`, `SegmentoDirigido`, `Circunferencia` y `Cuadrado`, disponibles en el fichero `FigurasGeometricas.cpp` disponible en `decsai`. Las tres primeras se han visto en clase de teoría. Con respecto al cuadrado, éste viene determinado por el punto correspondiente a la esquina inferior izquierda y por la longitud de cualquiera de sus lados (estos serán sus datos miembros). Supondremos que sólo representamos cuadrados cuya base es paralela al eje de las abscisas.

```
class Punto2D{
private:
   double abscisa;
   double ordenada;
public:
   ......
};
class SegmentoDirigido{
private:
   double x_1, y_1, x_2, y_2;
public:
   ......
};
class Circunferencia{
private:
      double centro_x;
      double centro_y;
      double radio;
public:
   ......
};
class Cuadrado{
private:
   double esquina_abscisa;
   double esquina_ordenada;
   double longitud;
public:
   ......
};
```

   Definid sobre la clase `Cuadrado` los siguientes métodos:

- Métodos para calcular el área y el perímetro del cuadrado.
- Obtener el punto central interior al cuadrado:

  `Punto2D Centro()`

  Para calcular las coordenadas basta sumar la mitad de la longitud del cuadrado a las coordenadas de la esquina inferior izquierda.
- Obtener la circunferencia inscrita al cuadrado (la que está por dentro):

  `Circunferencia CircunferenciaInscrita()`

  Esta circunferencia es la que tiene como centro el centro del cuadrado y como radio la mitad de la longitud del cuadrado.
- Obtener la circunferencia circunscrita al cuadrado (la que está por fuera):

  `Circunferencia CircunferenciaCircunscrita()`

  Esta circunferencia es la que tiene como centro el centro del cuadrado y como radio, la longitud del segmento que une el centro con la esquina inferior izquierda. Obtened la longitud creando el objeto de la clase `SegmentoDirigido` y a continuación llamad al método `Longitud`.
- Determinar si un cuadrado tiene mayor área que otro.

Complete el programa principal de prueba que se encuentra en el fichero `FigurasGeometricas.cpp`

Finalidad: Trabajar con el constructor de copia y con métodos que devuelven objetos. Dificultad Baja.

2. *(Examen Febrero 2012)* Sobre el ejercicio anterior, implemente un método para determinar si un cuadrado contiene a otro. Un cuadrado C_1 determinado por la esquina (x_1, y_1) y la longitud l_1 contiene a otro cuadrado C_2 dado por (x_2, y_2) y l_2 si se cumple que $x_2 >= x_1$ y $x_2 + l_2 <= x_1 + l_1$ y $y_2 >= y_1$ y $y_2 + l_2 <= y_1 + l_1$

   Finalidad: Pasar a un método de una clase un parámetro de la misma clase. Dificultad Baja.

3. Recupere la solución del ejercicio 28 de la Relación de Problemas IV. En este ejercicio se pedía construir una matriz *suavizada promedio*. Se quiere hacer lo mismo pero sobre una clase `MatrizCuadradaReales`, por lo que debe implementar el siguiente método:

   ```
   MatrizCuadradaReales SuavizadaPromedio(){
   ```

   En `decsai` se encuentra el código principal de la clase, así como de la clase `SecuenciaReales` y el programa principal. Complete el código implementando el método `SuavizadaPromedio` y la llamada desde el `main`.

4. *(Examen Febrero 2009)* Recupere la solución del ejercicio 30 de la Relación de Problemas IV (palabras similares).

   Sobre la clase `SecuenciaCaracteres`, definid un método que compruebe si la secuencia es similar a otra.

   Finalidad: Trabajar con métodos a los que se les pasa como parámetros objetos de la misma clase. Dificultad Baja.

5. *(Examen Septiembre 2014)* Recupere la solución del ejercicio 32 de la Relación de Problemas IV (Replace). Defina sobre la clase `SecuenciaCaracteres` el método `Replace` para que haga la tarea pedida. Tendrá que pasarle al método la posición inicial, el número de caracteres a eliminar y el objeto de la clase `SecuenciaCaracteres` conteniendo la secuencia de caracteres de reemplazo.

   Finalidad: Trabajar con métodos a los que se les pasa como parámetro y definen objetos locales de la MISMA clase.

   Dificultad Baja.

6. Recupere la solución del ejercicio 31 de la Relación de Problemas IV (elimina varios caracteres de una secuencia). Defina el método `EliminaVarios` sobre la clase `SecuenciaCaracteres` para que haga la tarea pedida. Tendrá que pasarle al método un objeto de la clase `SecuenciaEnteros` con los índices de las posiciones a eliminar.

   Finalidad: Pasar como parámetro a un método un objeto de otra clase. Dificultad Media.

7. Se quiere calcular la moda de una secuencia de caracteres, es decir, el carácter que más veces se repite. Por ejemplo, si la secuencia fuese

   ```
   {'l','o','s',' ','d','o','s',' ','c','o','f','r','e','s'}
   ```

   los caracteres que más se repiten son `'o'` y `'s'` con un total de 3 apariciones. La moda sería cualquiera de ellos, por ejemplo, el primero encontrado `'o'`. Sobre la clase `SecuenciaCaracteres`, se pide construir el método `Moda` que devuelva un `struct` del tipo:

   ```
   struct FrecuenciaCaracter{
      char caracter;
      int  frecuencia;
   }
   ```

   en el que el campo `caracter` contendrá el carácter en cuestión (`'o'`) y en el campo `frecuencia` el conteo de la moda (3).

   Finalidad: Usar como dato local de un método un vector y devolver un `struct`. *Dificultad Baja.*

8. En el primer apartado del ejercicio 8 de la relación de problemas III se pedía eliminar los repetidos de un objeto de la clase `SecuenciaCaracteres` usando como dato auxiliar local un vector clásico con los elementos que no estuviesen repetidos. Recuperad la solución vista en clase y resolved este mismo apartado usando como dato auxiliar local un objeto de la propia clase `SecuenciaCaracteres`

9. *(Examen Septiembre 2014)* Sobre la clase `SecuenciaCaracteres` implemente el algoritmo **Counting Sort** para ordenar sus valores. El método no modificará las componentes del vector privado sino que debe construir una secuencia nueva y devolverla. El algoritmo funciona de la siguiente forma:

   - Calculad los caracteres mínimo y máximo del vector. Por ejemplo, si el vector contiene

     ```
     c b b a b c c a g c b g c
     ```

     el mínimo es `'a'` y el máximo `'g'`.
   - Construid un vector auxiliar de frecuencias con los conteos de todos los caracteres que hay entre el mínimo y el máximo. Con el ejemplo anterior, el vector de conteos será

     ```
     2 4 5 0 0 0 2
     ```

     que corresponden a las frecuencias de las letras que hay entre `'a'` y `'g'`.
   - Recorrer el vector de frecuencias almacenando cada carácter tantas veces como indique su frecuencia (2 veces el `'a'`, cuatro veces el `'b'`, etc)

     ```
     a a b b b b c c c c c g g
     ```

   Haced lo mismo pero parametrizando el método `CountingSort` para que ordene sólo los valores de la secuencia que hay entre un carácter `izquierda` y otro carácter `derecha`. Por ejemplo, si `izquierda = 'b'` y `derecha = 'g'` el resultado sería:

   ```
   b b b b c c c c c g g
   ```

   Finalidad: Trabajar con métodos con vectores locales y devolviendo un objeto de la misma clase. Dificultad Media.

10. Definid la clase `MatrizRectangularEnteros` usando una matriz de doble corchete como dato miembro privado.

    ```
    int matriz_privada[MAXIMO_FILAS][MAXIMO_COLUMNAS];
    ```

    Definid métodos para:

    a) Obtener el número de filas y columnas utilizadas, así como el dato que haya en una fila y columna.

b) Devolver una fila completa como un objeto de la clase `SecuenciaEnteros`.

c) Añadir una fila entera. La fila será un objeto de la clase `SecuenciaEnteros`.

d) Comprobar si es igual a otra matriz.

e) Obtener la traspuesta.

f) Comprobar si es simétrica. Hacedlo primero calculando la traspuesta de la matriz y viendo si es igual a su simétrica, usando los métodos anteriores.

Hacedlo también comprobando directamente si cada componente es igual a su *simétrica* y parando el recorrido en cuanto encuentre una componente que no lo verifique.

g) Multiplicar dos matrices.

Finalidad: Trabajar con matrices. Dificultad Baja.

11. Sobre el ejercicio anterior, construid un método que busque la fila de la matriz que más se parezca a una secuencia de enteros, a la que llamaremos `referencia`. La similitud entre dos secuencias $x = (x_1 \cdots x_p)$ e $y = (y_1 \cdots y_p)$ vendrá dada por la distancia euclídea entre ambas:

$$dist(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

Además, la búsqueda solo se hará sobre las filas de la matriz enumeradas en una segunda secuencia llamada `filas_a_comparar`.

Por ejemplo, dada la matriz M (7×4),

$\rightarrow$	3	1	0	8
	4	5	1	5
$\rightarrow$	5	7	1	7
	7	9	6	1
$\rightarrow$	4	9	5	5
$\rightarrow$	2	8	2	2
	7	3	2	5

y las secuencias `referencia` $= 2, 8, 1, 1$ y `filas_a_comparar` $= 0, 2, 4, 5$, el programa deberá encontrar 5 como la fila más cercana a `referencia`. En el dibujo anterior se han marcado con una flecha las filas indicadas por `filas_a_comparar`.

Finalidad: Trabajar con matrices. Dificultad Media.

12. *(Examen Septiembre 2013)* **Sudoku** es un juego muy popular que consiste en rellenar una cuadrícula de 9×9 celdas que está dividida en subcuadrículas de 3×3 (denominadas *regiones*) con cifras del 1 al 9. Un sudoku se considera resuelto si verifica que:

- En cada fila aparecen todos los números del 1 al 9 (sin repetir)
- En cada columna aparecen todos los números del 1 al 9 (sin repetir)
- En cada región aparecen todos los números del 1 al 9 (sin repetir)

Realizar un programa que lea todos los elementos de un sudoku y determine si está resuelto o no. Un ejemplo de sudoku resuelto es el siguiente:

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Implemente la solución con una clase `Sudoku` con un dato miembro matriz de doble corchete y añada un programa principal de prueba.

Finalidad: Trabajar con matrices. Dificultad Media.

13. *(Examen Febrero 2013)* Se quiere trabajar con una tabla de datos en el que todas las filas tienen el mismo número de columnas y los datos son de tipo `int`. Esta clase se llamará `ImagenBlancoNegro` y contendrá valores enteros que representan un nivel de gris (0 sería negro y 255 blanco). Se supone que todos los valores deben ser positivos aunque por problemas de captación y registro algunos de ellos son negativos. Es preciso corregir estos valores erróneos y se propone sustituirlos por el valor promedio de sus **ocho** vecinos más cercanos espacialmente (arriba, abajo, izquierda, derecha y esquinas). Debe considerar que entre estos vecinos pudiera haber valores negativos, y en este caso **no** intervendrán en el cálculo del valor promedio:

    > Si hubiera un sólo valor negativo en la vecindad, se sumarán los valores de los 7 vecinos válidos y la suma se dividirá entre 7. Si hubiera dos valores negativos en la vecindad, se sumarán los valores de los 6 vecinos válidos y la suma se dividirá entre 6. . . . Si no hubiera ningún valor válido, se sustituirá por un cero.

    Implemente un método para que dada una imagen, devuelva **otra** imagen corregida. La imagen original **no** se modifica.

    Para la implementación debe considerar:

    a) El algoritmo debe ser simple y claro.

b) Para simplificar el problema, las casillas de los bordes **no** se modifican, aunque **sí** se usan para efectuar las correcciones oportunas. En definitiva, la primera y la última fila así como la primera y la última columna son iguales entre la matriz original y la corregida.

Cread un programa principal de prueba.

Finalidad: Trabajar con matrices. Dificultad Media.

14. *(Examen Septiembre 2012)* Definid la clase `VectorParejasCaracterEntero` que permite almacenar un conjunto de parejas de la forma `(carácter, entero)`. Cada pareja será un struct `ParejaCaracterEntero` con un campo de tipo carácter y otro campo de tipo entero.

```
struct ParejaCaracterEntero{
   char caracter;
   int veces;
}
```

Se pide crear un método de la clase `SecuenciaCaracteres` al que se le pasará como parámetro un objeto de la clase `VectorParejasCaracterEntero` para que borre cada uno de los caracteres que aparecen en el vector de parejas, tantas veces como indique el entero correspondiente. Por ejemplo:

```
Borrar ({(a,1),(b,2)}) en {a,b,a,b,c,a,b,d,a}
-> {a,c,a,b,d,a}
```

Finalidad: Trabajar con vectores de struct. Dificultad Media.

15. *(Examen Septiembre 2013)* Se quiere almacenar el nombre de un alumno junto con las notas que ha sacado en varias asignaturas. El número de asignaturas puede variar de un alumno a otro y las notas son de tipo de dato entero. Con estos datos, se quiere construir un conjunto de alumnos

```
Ana de Gober, (9,7,8,9)
Sergio García, (3,4,2)
David Rodriguez, (5,4)
```

Defina la clase `Alumnos` que contendrá los siguientes datos miembro:

- Un vector de `SecuenciaCaracteres` con los nombres de los alumnos: `[ {Ana de Gober} , {Sergio García} , {David Rodriguez} ]`
- Un vector de `SecuenciaEnteros` con las notas de cada alumno: `[ {9,7,8,9} , {3,4,2} , {5,4} ]`

Añada métodos para:

- Obtener el número total de alumnos
- Obtener el nombre de un alumno. Al método se le pasará un índice de componente y devolverá un objeto `SecuenciaCaracteres`.
- Obtener todas las notas de un alumno. Al método se le pasará un índice de componente y devolverá un objeto `SecuenciaEnteros`.
- Obtener los alumnos cuyo nombre contenga una determinada cadena de caracteres. Al método se le pasará un objeto `SecuenciaCaracteres` y devolverá un objeto `SecuenciaEnteros` con las posiciones correspondientes.
- Ordenar los datos de menor a mayor según la media aritmética de sus calificaciones. Con los datos del anterior ejemplo, la nota media de `Ana de Gober` sería 8.25, la de `Segio García` 3 y la de `David Rodriguez` 4.5 , por lo que los datos ordenados quedarían como sigue:

  ```
  Sergio García, (3,4,2)
  David Rodriguez, (5,4)
  Ana de Gober, (9,7,8,9)
  ```

- Cread un programa principal sencillo de prueba.

Finalidad: Trabajar con vectores de objetos. *Dificultad Media.*

16. En las transparencias se enfatiza que *las tareas necesarias para realizar las operaciones de E/S de los datos de un objeto, se realizarán en clases específicas que implementen dichas responsabilidades.*

    Vamos a aplicarlo sobre la clase `SecuenciaCaracteres`. Para ello, vamos a crear sendas clases:

    - La clase `ImpresorSecuenciaCaracteres` servirá para imprimir los caracteres de un objeto de la clase `SecuenciaCaracteres`. Si la secuencia contiene los caracteres `h o l a`, por ejemplo, en pantalla saldrá lo siguiente:

      ```
      {h,o,l,a}
      ```

      En general, se dará la posibilidad de delimitar los caracteres con otros símbolos que no sean `{ } ,`
      La clase `ImpresorSecuenciaCaracteres` contendrá el siguiente método:

      ```
      void Imprime (SecuenciaCaracteres secuencia_a_imprimir)
      ```

    - La clase `LectorSecuenciaCaracteres` para leer los datos de un objeto de la clase `SecuenciaCaracteres`. La lectura de datos parará cuando se llegue a un *terminador*, que será un carácter especial.
      La clase `LectorSecuenciaCaracteres` contendrá al menos el siguiente método:

      ```
      SecuenciaCaracteres Lee()
      ```

que será el encargado de hacer la lectura de los datos y construir un objeto de la clase `SecuenciaCaracteres`.

Cread un programa principal que vaya leyendo datos de un fichero. El fichero contendrá al principio el número de filas de caracteres que hay y a continuación las filas. Cada fila será una serie de caracteres con un punto al final (éste es el terminador de la fila).

```
3
Esto es una fila.
Esta es otra.
Tercera y última fila.
```

Para realizar la lectura se usará un único objeto de la clase `LectorSecuenciaCaracteres`. Cada fila se almacenará en un objeto de la clase `SecuenciaCaracteres`. Cada vez que leamos una fila la añadiremos a un objeto de la clase `Texto`. Utilizad la implementación incluida en las transparencias en el apartado *Tabla dentada usando un vector de objetos*, bajo el nombre `Texto_vs2`. Este objeto `Texto` habrá que definirlo en la función `main`.

Una vez leídas todas las filas, crearemos un objeto de la clase `ImpresorSecuenciaCaracteres` y lo usaremos para imprimir por pantalla todas las filas.

Finalidad: Trabajar con clases cuya responsabilidad es realizar las tareas de E/S. Dificultad Media.

17. Sobre la base del ejercicio 16, construid una clase `ImpresorTexto` para que imprima un objeto de la clase `Texto`.

    Finalidad: Trabajar con clases cuya responsabilidad es realizar las tareas de E/S. Dificultad Baja.

18. Sobre la base del ejercicio 15 (conjunto de alumnos), definid una clase `ImpresorAlumnos` para que imprima un objeto de la clase `Alumnos`. Defina también una clase `LectorAlumnos` para que construya un objeto `Alumnos` a partir de los datos de un fichero de la forma que crea más conveniente. Cread un programa principal de prueba.

    Finalidad: Trabajar con clases cuya responsabilidad es realizar las tareas de E/S. Dificultad Baja.

19. *(Examen Septiembre 2013)* Queremos saber si dos círculos intersecan. Para ello, basta ver que la distancia entre sus centros debe ser menor o igual que la suma de sus radios (por tanto, supondremos que dos círculos concéntricos se intersecan). Se pide construir las **clases y métodos** necesarios para resolver este problema, teniendo en cuenta lo siguiente:

- Debe incluir la definición de los datos miembros y la implementación del constructor de todas las clases que necesite.
- Debe incluir las cabeceras de los métodos necesarios para resolver el problema, pero no tiene que incluir la implementación de dichos métodos.
- Debe incluir la implementación del método que comprueba la intersección, pero no tiene que incluir la implementación de los métodos invocados dentro de él.

20. *(Examen Septiembre 2012)* Definid la clase `ConjuntoOrdenado` para que permita almacenar una secuencia **ordenada** de números enteros **sin repetidos**. Definid métodos para:

    - Añadir un entero (de forma ordenada y sin almacenar repetidos).
    - Calcular la unión con otro conjunto. En la unión se deben incluir los elementos que estén en cualquiera de ellos.
    - Calcular la intersección con otro conjunto. En la intersección se deben incluir los elementos que sean comunes a ambos conjuntos.

    Cread un programa principal de prueba.

    Finalidad: Trabajar con métodos a los que se les pasa como parámetro y devuelven objetos de la misma clase. Dificultad Media.

21. Recuperad la solución al problema 12 del ComeCocos de la Relación de Problemas IV. Sobre la clase `CaminoComeCocos`, añadidle un método que compruebe si el conjunto de movimientos de un camino contiene a los movimientos de un segundo camino que se pasará como parámetro al método. Debe respetarse el orden en el que aparecen los movimientos, pero no tienen por qué estar consecutivos. Por ejemplo, el camino `{'s','s','b','i','d','d'}` contiene al camino `{'s','i'}` pero no al camino `{'i','s'}`. Puede usarse la sobrecarga del método `find` de la clase `string` a la que se le pasa como parámetro la posición inicial desde la que se realiza la búsqueda:

    ```
    string cadena = "Hola, soy yo";
    int pos;
    pos = cadena.find('o',2);     // 7 -> Primera ocurrencia de 'o'
                                  // a partir de la posición 2
    pos = cadena.find('o',1);     // 1
    pos = cadena.find('H',1);     // -1 No encontrado.
    ```

    Finalidad: Pasar a un método de una clase un parámetro de la propia clase. Dificultad Baja.

22. *(Examen Septiembre 2012)* Se quiere desarrollar una aplicación para automatizar la realización de exámenes tipo test. El software incluirá una clase `Examen` que debe almacenar: el nombre de la asignatura, la lista de enunciados de las preguntas (cada

enunciado es una cadena de caracteres de tipo `string`) y la lista de respuestas correctas para cada pregunta (cada respuesta es un carácter). Implementa la clase junto con los siguientes métodos:

- Un constructor que inicialice un objeto de tipo `Examen` dando el nombre de la asignatura y con la lista de preguntas vacía.
- Un método `NuevaPregunta` que reciba un enunciado y la respuesta correcta y que los añada a la lista de preguntas del examen. Cada nueva pregunta siempre se añade al final de la lista.
- Un método `NumPreguntas` que devuelva el número de preguntas de que consta el examen.
- Un método `Enunciado` que devuelva el enunciado de la pregunta i-ésima.
- Un método `Respuesta` que devuelva la respuesta de la pregunta i-ésima.

A continuación, se pide realizar un programa que permita evaluar a una serie de alumnos utilizando la clase `Examen`. El programa comenzará creando un objeto de tipo `Examen` y dándole contenido, es decir, leyendo las preguntas y respuestas correctas desde la entrada estándar y almacenándolas.

Una vez leído el examen se procederá a la evaluación de un número de alumnos dado desde la entrada estándar. Para ello el programa le mostrará las preguntas del examen a cada alumno y leerá sus respuestas. Al finalizar cada alumno la prueba, el programa le dirá su nota de acuerdo a los siguientes criterios:

- Por cada pregunta sin responder se suman 0 puntos.
- Por cada respuesta correcta se suma 1 punto.
- Por cada respuesta incorrecta se resta 1 punto.
- La nota final estará en el intervalo [0, 10]. Un 10 significa que ha respondido y acertado todas las preguntas. Si la calificación es negativa se sustituye por cero.

No es necesario almacenar las notas de los alumnos ya que se pueden ir mostrando al terminar cada uno de ellos la prueba. Además, se pueden añadir nuevos métodos a la clase `Examen` si lo considera oportuno.

23. *(Examen Septiembre 2009)* Sobre la clase `SecuenciaCaracteres`, añadid un método que determine si dicha secuencia de caracteres *C1* contiene a otra secuencia *C2* en el mismo orden (no tienen que estar consecutivos) y de forma *cíclica*. Para que se cumpla este criterio se deben satisfacer las siguientes condiciones

    - Todos los caracteres de *C2* deben estar en *C1*
    - Deben estar en el mismo orden aunque no de forma consecutiva
    - Si durante la búsqueda se ha llegado al final de la secuencia *C1*, se debe proseguir la búsqueda por el inicio de *C1*, pero sin sobrepasar la posición en la que hubo la primera concordancia.

Se muestran algunos ejemplos en los que la secuencia *C1* contiene a *C2*:

- *C1* = xzayobnmcpwqdfg *C2* = abcd
- *C1* = ftkcpxqdhjzaxqoblki *C2* = abcd
- *C1* = tzsbluyclpygdmngrafvc *C2* = abcd

Hay que destacar que la primera letra de *C2* a buscar en *C1* podría estar en cualquier sitio. Por ejemplo, para el siguiente caso:

- *C1* = bghcjadxak *C2* = abcd

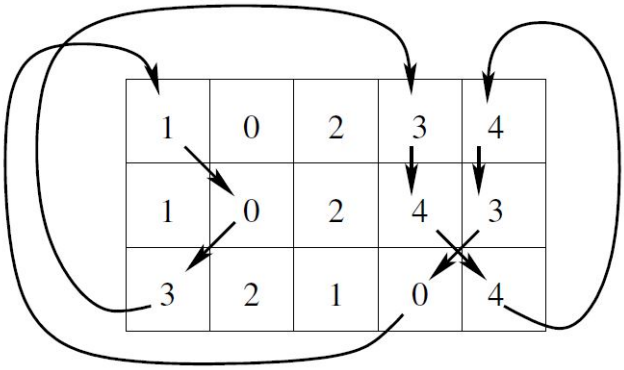
podemos ver que a partir de la primera a de *C1* no podemos encontrar *C2* de forma cíclica, aunque sí lo podemos hacer a partir de la segunda a de *C1*.

También puede darse el caso de que *C1* no contenga a *C2* de forma cíclica aunque incluya todas sus letras, como muestra el siguiente ejemplo:

- *C1* = tzsbluyclpcaygdmxngrfvc *C2* = abcd

24. Recuperad la clase `MatrizRectangularEnteros` (ejercicio 10 de esta relación). Definid un método que ordene las columnas en función de la media aritmética de cada una de ellas. Así pues, después de aplicar el método, la media de la columna `i` será menor o igual que la media de la columna `j` para cualquier `i < j`.

25. *(Examen Septiembre 2005)* Recuperad la clase `MatrizRectangularEnteros` (ejercicio 10 de esta relación). Supongamos que cada casilla representa un *enlace* a una casilla de la siguiente fila. Dicho enlace es únicamente un entero que indica un índice de columna de la siguiente fila, de forma que una casilla de la última fila enlaza con otra casilla de la primera fila. Suponiendo que la matriz tiene datos correctos (no hay valores imposibles de columnas y todos los valores de una fila son distintos), se pide construir un método que calcule cuantos *ciclos* hay, es decir, cuantos caminos hay que empiecen en un valor de la primera fila y siguiendo los enlaces, se llegue de nuevo al mismo valor de la primera fila. Por ejemplo, en la matriz de abajo hay un total de dos ciclos (uno pintado con flechas y otro más sin pintar).



26. *(Examen Febrero 2009)* Recuperad la clase `SopaLetras` vista en clase de teoría en el apartado *Tabla rectangular usando una matriz*. Queremos definir una **medida de similitud** entre dos sopas de letras M_1 y M_2 cuadradas $n \times n$ de la siguiente forma:

$$S(M_1, M_2) = |M_1 \cap M_2| + n_p$$

donde n_p es el número de posiciones (i, j) ($i = 1, \cdots, n$ y $j = 1, \cdots, n$) en las que M_1 y M_2 tienen el mismo elemento. Por ejemplo, dadas las siguientes sopas de letras:

$$M_1 = \begin{vmatrix} \text{x} & \text{i} \\ \text{n} & \text{k} \end{vmatrix} \quad M_2 = \begin{vmatrix} \text{n} & \text{i} \\ \text{q} & \text{p} \end{vmatrix}$$

entonces $S(M_1, M_2) = 2 + 1 = 3$, ya que tienen dos caracteres en común (i y n) y una posición ($i = 1$ y $j = 2$) en la que ambas sopas tienen el mismo carácter. Definid un método que implemente el cómputo de esta medida de similitud.

27. *(Examen Febrero 2013)* Queremos representar un conjunto de equipos de tenis de mesa participantes en un torneo. Únicamente queremos almacenar el nombre de cada uno de ellos en un `string`. Se desea construir el conjunto de todos los emparejamientos posibles. Por ejemplo, si se parte del conjunto

    { Albolote, Motril, Baza, La Zubia }

    quiere construirse el siguiente conjunto de emparejamientos:

    {{Albolote, Motril} , {Albolote, Baza} , {Albolote, La Zubia} , {Motril, Baza} , {Motril, La Zubia} , {Baza, La Zubia}}

    Para resolver este problema definiremos la clase `SecuenciaString` para representar una secuencia de datos de tipo `string` y crearemos en el programa principal un objeto `equipos` de esta clase. Debe definir también la clase `SecuenciaParejasString` para poder representar un conjunto arbitrario de parejas de `string`. Lo más fácil es utilizar como dato miembro privado un vector clásico de corchetes en el que cada componente es un registro del siguiente tipo:

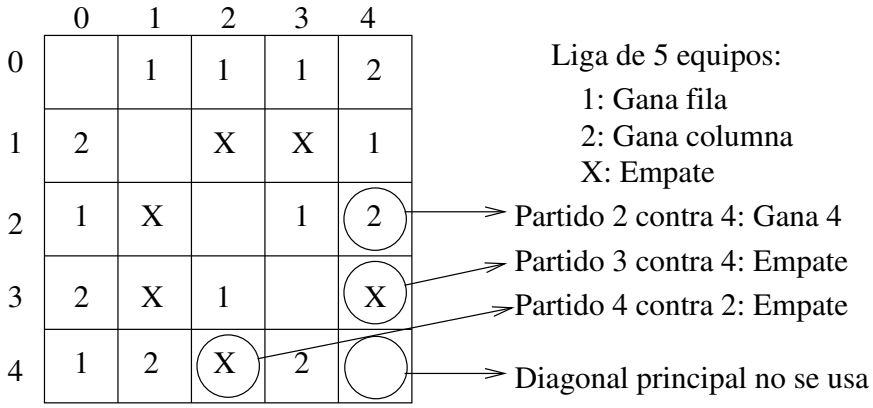
```
struct ParejaString{
   string cadena_izda;
   string cadena_dcha;
};
```

    Tenga en cuenta que el número de equipos puede ser cualquiera (no sólo 4 como en el ejemplo)

    Finalidad: Trabajar con un vector de struct. Dificultad Baja.

28. *(Examen Febrero 2013)* Para gestionar un campeonato de n equipos se utiliza una matriz de tamaño $n \times n$ (el máximo número de equipos que se permite es 20, pero

podría haber menos de 20). En cada posición de esta matriz se pueden almacenar tres posibles valores (`'1'`, `'X'`, `'2'`). La fila f y columna c contendrá un valor correspondiente al partido que enfrenta al equipo f con el c, de forma que si vale `'1'` indica que ha ganado f, si vale `'X'` han empatado y si vale `'2'` ha ganado c.



Observe que dados dos equipos (m, n) habrá dos partidos: uno en el que se enfrentan m y n y el recíproco, de n con m. Además, el valor de la diagonal no se usa, ya que no existe el partido n contra n.

Se pide crear una clase `Liga` para manejar esta información. La clase debe contener un método que construya una secuencia de enteros con los resultados finales de la liga. Estos resultados contabilizan, para cada equipo, los puntos obtenidos: la componente 0 contendrá los puntos del primer equipo, la componente 1 los del segundo y así sucesivamente. Tened en cuenta que una victoria implica 3 puntos, un empate 1 punto, y una derrota 0 puntos.

- Para representar la matriz de datos, utilice el tipo de tabla que considere más adecuado (según lo visto en las transparencias)
- Cread también la clase `GeneradorLiga` con un método que permita leer los datos de los resultados de los equipos desde un fichero y construya el objeto `Liga`.

```
class GeneradorLiga{
public:
   Liga Lee(){
      ......
   }
};
```

- Cread un programa principal que lea los datos de la liga, obtenga los puntos y los imprima por pantalla.

29. Vamos a usar una clase para generar números enteros aleatorios entre un mínimo y un máximo con la siguiente interfaz pública:

```
MyRandom
   + MyRandom(int minimo, int maximo)
   + int Next()
   + int Min()
   + int Max()
```

Puede usarse cualquiera de las implementaciones que vienen a partir de la página RP-V.16 (copie y pegue el código de dicha clase en su programa).

Para generar 10 números aleatorios entre 4 y 7, por ejemplo, bastaría hacer lo siguiente:

```
MyRandom generador_aleatorio(4, 7);

for (int i=0; i<10; i++)
   cout << generador_aleatorio.Next();
```

Hay que destacar lo siguiente:

- Cada llamada a `generador_aleatorio.Next()` genera un valor aleatorio (entre 4 y 7 en el ejemplo)
- Los valores generados pueden repetirse antes de que se hayan generado todos los posibles valores. Por lo tanto, una posible secuencia de números generados podría ser la siguiente: `5 4 5 6 4 7 4 5`

Se pide crear la clase `GeneradorPermutaciones` para generar permutaciones aleatorias de un conjunto de enteros entre un valor mínimo y un valor máximo. La clase tendrá un único método con la siguiente cabecera:

```
Permutacion Genera(int primero, int ultimo)
```

dónde la clase `Permutacion` es la vista en el problema 13 de la relación de problemas IV. Por ejemplo, si mínimo = 1 y máximo = 6, una permutación válida sería `{3,1,6,4,5,2,3}`. Como puede observarse, no pueden aparecer elementos repetidos y deben estar todos los valores entre 1 y 6.

Apéndice: Clase MyRandom

Proporcionamos dos posibles implementaciones de la clase `MyRandom` para generar números enteros aleatorios entre un mínimo y un máximo.

Opción 1. Siguiendo el nuevo estándar de C++ 11.

```
#include <random>  // para la generación de números pseudoaleatorios
#include <chrono>  // para la semilla

class GeneradorAleatorioEnteros{
private:
   mt19937 generador_mersenne;   // Mersenne twister
   uniform_int_distribution<int>  distribucion_uniforme;
public:
   GeneradorAleatorioEnteros()
      :GeneradorAleatorioEnteros(0, 1){
   }
   GeneradorAleatorioEnteros(int min, int max){
      auto semilla =
         chrono::high_resolution_clock::now().time_since_epoch().count();
      generador_mersenne.seed(semilla);
      distribucion_uniforme = uniform_int_distribution<int> (min, max);
   }
   int Siguiente(){
      return distribucion_uniforme(generador_mersenne);
   }
};
```

Opción 2. A la antigua usanza, para aquellos compiladores que no proporcionen la biblioteca `random`.

```
#include <cstdlib>    // Hay que incluir estas bibliotecas
#include <ctime>

class MyRandom {
private:
    int minVal;
    int maxVal;

    void InitMyRandom (void)
    {
        time_t t;
        srand ((int) time(&t));     // Inicializa el generador
                                    // con el reloj del sistema
    }
public:
    MyRandom (int el_minimo, int el_maximo) :
        minVal(el_minimo), maxVal(el_maximo)
    {
        InitMyRandom();

        int no_lo_uso = Next(); // desecho el primero
    }
    int Next()
    {
        int rango = (maxVal - minVal)+1;
        int v1 = rango * (rand() / (RAND_MAX*1.0));
        int v2 = minVal + (v1 % rango);

        return v2;
    }
    int Min()
    {
        return minVal;
    }
    int Max()
    {
        return maxVal;
    }
```