Fundamentos de Programación
Relación de ejercicios 4 (clases)

1. (Recta) En este ejercicio se plantean varias modificaciones. Debe entregar un fichero cpp por cada uno de los apartados. Se desea implementar una clase `Recta` para representar una recta en el plano. Una recta viene determinada por tres coeficientes A, B, C, de forma que todos los puntos (x, y) que pertenecen a la recta verifican lo siguiente (ecuación general de la recta):

$$Ax + By + C = 0$$

   a) Definición de la clase y creación de objetos. Defina la clase Recta. En este apartado utilice únicamente datos miembro públicos. Cree un programa principal que haga lo siguiente:
   - Defina dos objetos de la clase `Recta`.
   - Lea seis reales desde teclado.
   - Le asigne los tres primeros a los coeficientes de una recta y los otros tres a la segunda recta.
   - Calcule e imprima la pendiente de cada recta aplicando la fórmula:

$$pendiente = -\frac{A}{B}.$$

   b) Métodos públicos. En vez de calcular la pendiente en el programa principal, vamos a ponerlo como un método de la clase y así lo reutilizaremos todas las veces que necesitemos. Añada un método para el cálculo de la pendiente y modificad el main para tener en cuenta este cambio. ¿Añadimos pendiente como dato miembro de la recta? La respuesta es que no ¿Por qué? Añadir también los siguiente métodos:
   - Obtener la ordenada (y) dado un valor de abscisa x, aplicando la fórmula:

$$\frac{-C - xA}{B}$$

   - Obtener la abscisa (x) dado un valor de ordenada y, aplicando la fórmula:

$$\frac{-C - yB}{A}$$

   En la función `main` leed un valor de abscisa e imprimir la ordenada según la recta y leed un valor de ordenada e imprimid la abscisa que le corresponde. Hacedlo sólo con la primera recta.

   c) Datos miembro privados. Cambie ahora los datos miembro públicos y póngalos privados. Tendrá que añadir métodos para asignar y ver los valores de los datos miembro. Añada métodos para asignar un valor a cada uno de los tres datos miembro. Modifique el `main` para tener en cuenta estos cambios. A partir de ahora, todos los ejercicios deben resolverse utilizando únicamente datos miembro privados.

   d) Política de acceso a los datos miembros. En vez de usar un método para asignar un valor a cada dato miembro, defina un único método `SetCoeficientes` para asignar los tres a la misma vez. Observad que los métodos permiten definir la política de acceso a los datos miembro. Si tengo previsto cambiar por separado los coeficientes de la recta, usaré métodos de asignación individuales. En caso contrario, usaré un único método que modifique a la misma vez todos los datos miembro. Incluso pueden dejarse en la clase ambos tipos de métodos para que así el cliente de la clase pueda usar los que estime oportunos en cada momento. Por ahora, mantenga únicamente el método de asignación en bloque `SetCoeficientes`.

e) Constructor. Modifique el programa principal del último apartado e imprima los valores de los datos miembros de una recta, antes de asignarles los coeficientes. Mostrará, obviamente, un valor indeterminado. Para evitar este problema, añada un constructor a la recta para que el objeto esté en un estado válido en el mismo momento de su definición. El constructor deberá tener como parámetros, obligatoriamente, los tres coeficientes de la recta. Tendrá que modificar convenientemente el `main` para tener en cuenta este cambio.

f) Política de acceso a los datos miembro. Suprima ahora el método `SetCoeficientes`. De esta forma, una vez creado el objeto (pasándole los datos apropiados en el constructor) ya no podremos modificar los datos miembro. Esto es útil en aquellas situaciones en las que no queremos permitir que el estado del objeto cambie, una vez se ha creado.

g) Métodos privados. Añada un método privado que nos indique si los coeficientes son correctos, es decir, A y B no pueden ser simultáneamente nulos. Llame a este método en el constructor y en el método `SetCoeficientes` y realice las operaciones que estime oportuno en el caso de que los coeficientes pasados como parámetros no sean correctos.

2. (DepositoSimulacion) Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 13 y 14 de la relación de ejercicios 2. Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

   a) Calcular el capital que se obtendrá al cabo de un número de años,

   b) Calcular el número de años que deben pasar hasta obtener el doble de la cantidad inicial.

   A la hora de diseñar la clase, tendremos que analizar cuestiones como:

   a) ¿Cuáles son sus datos miembro? Parece claro que el capital y el interés sí lo serán ya que cualquier operación que se nos ocurra hacer con un objeto de la clase `DepositoSimulacion` involucra a ambas cantidades. ¿Pero y el número de años?

   b) ¿Qué constructor definimos?

   c) ¿Queremos modificar el capital y el interés una vez creado el objeto? ¿Queremos poder modificarlos de forma independiente?

   d) ¿Hay alguna restricción a la hora de asignar un valor al capital e interés?

   e) ¿Es mejor un método para calcular el número de años hasta obtener el doble de la cantidad inicial, o por el contrario es mejor un método para calcular el número de años hasta obtener una cantidad específica?

   Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

3. (Complejo) Implementar una clase para representar números complejos. Debe incluir, al menos, operaciones para sumar, restar y multiplicar complejos, además de otros métodos para asignar un valor al número complejo, para obtener su parte real y para obtener su parte imaginaria. Hacer un programa de ejemplo que muestre cómo se ejecuta cada método.

4. (Fecha) Implementa una clase para representar una fecha del calendario (Por ejemplo: 10/02/2010). Implemente métodos para asignar una fecha, para saber si la fecha es válida, para actualizar la fecha a la del día siguiente, y para actualizar la fecha a la del día anterior. Hacer un programa de ejemplo que muestre cómo se ejecuta cada método.

5. (Libro) Implemente una clase para representar un libro con los campos Autor, Título, Editorial, Año. Implemente métodos para asignar un libro, mostrar sus campos por pantalla, devolver cada campo y comparar dos libros. Haga un programa de ejemplo que muestre cómo se ejecuta cada método.

6. (Conjunto) Se desea implementar una clase `Conjunto` que represente un conjunto de números enteros usando los siguientes datos miembro:

```
class Conjunto{
   private:
      int num_elem;
      int elementos[MAXELEM];
   .....
};
```

Los elementos del conjunto se almacenan en un vector. La memoria reservada viene dada por una constante entera `MAXELEM` (el máximo número posible de enteros almacenados) y `num_elem` indica el cardinal del conjunto, es decir, el número de elementos que actualmente forman parte del mismo (siempre menor que `MAXELEM`). La constante `MAXELEM` puede ser una constante global definida fuera de la clase, o bien, definida dentro de la clase como dato miembro. En este segundo caso, hay que definirla como `static`, esto es, si, por ejemplo, definimos un tamaño máximo de 100, `static const int MAXELEM = 100`. Además, el vector con los elementos se encuentra ordenado de forma ascendente, y después de cada modificación sobre el vector se debe conservar dicha ordenación. Se pide implementar los siguientes métodos miembro para:

- inicializar una variable de tipo de dato `Conjunto` a un conjunto con 0 elementos.
- inicializar una variable de tipo de dato `Conjunto` utilizando un vector (array) de enteros (no necesariamente ordenados).
- para determinar si un entero se encuentra en un conjunto.
- para incorporar un nuevo elemento a un conjunto (en un conjunto no hay elementos repetidos).
- eliminar un elemento de un conjunto.
- calcular la unión con otro conjunto.
- calcular la intersección con otro conjunto.

¿Y si quisiéramos que las funciones de unión e intersección no fueran funciones miembro? Implementa funciones externas `union2` e `interseccion2` para obtener esta funcionalidad.

7. (EnteroLargo) Diseña e implementa una clase `EnteroLargo`, que permita realizar operaciones aritméticas con un número de dígitos arbitrario (hasta cierto tope que se determina a priori, claro está). Para ello, considera que un número entero se puede representar como una secuencia de elementos. Implementa como ejemplos las operaciones de menor, mayor, igual, distinto, menor o igual, mayor o igual, asignación, suma y resta.

8. (Nomina) Se quiere construir una clase `Nomina` para realizar la funcionalidad descrita en el ejercicio 5 de la relación de ejercicios 1 sobre la nómina del fabricante y diseñador. Cread los siguientes programas (un fichero por cada uno de los apartados):

   a) Suponed que sólo gestionamos la nómina de una empresa en la que hay un fabricante y tres diseñadores. Los salarios brutos se obtienen al repartir los ingresos de la empresa, de forma que el diseñador cobra el doble de cada fabricante. El programa leerá el valor de los ingresos totales y calculará los salarios brutos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

   b) Supongamos que se aplica una retención fiscal y que ésta es la misma para los fabricantes y el diseñador. En el constructor se establecerá el porcentaje de retención fiscal (de tipo

`double`) y posteriormente no se permitirá que cambie, de forma que todas las operaciones que se hagan serán siempre usando la misma retención fiscal. Los salarios netos se obtienen al aplicar la retención fiscal a los salarios brutos (después de repartir los ingresos totales de la empresa):

$$SalarioNeto = SalarioBruto - SalarioBruto * RetencionFiscal/100.0$$

El programa leerá el valor de los ingresos totales y la retención fiscal a aplicar y calculará los salarios brutos y netos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

c) Supongamos que gestionamos las nóminas de varias sucursales de una empresa. Queremos crear objetos de la clase `Nomina` que se adapten a las características de cada sucursal:

- En cada sucursal hay un único diseñador pero el número de fabricantes es distinto en cada sucursal. Por tanto, el número de fabricantes habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.
- La forma de repartir el dinero es la siguiente: el diseñador se lleva una parte del total y el resto se reparte a partes iguales entre los fabricantes. En los apartados anteriores, por ejemplo, la parte que se llevaba el diseñador era 2/5 y el resto (3/5) se repartía entre los tres fabricantes. La parte que el diseñador se lleva puede ser distinta entre las distintas sucursales (2/5, 1/6, etc), pero no cambia nunca dentro de una misma sucursal. Por tanto, el porcentaje de ganancia (2/5, 1/6, etc) habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.
- Las retenciones fiscales de los fabricantes y diseñador son distintas. Además, se prevé que éstas puedan ir cambiando durante la ejecución del programa. Por lo tanto, no se incluirán como parámetros en el constructor.

El programa leerá los siguientes datos desde un fichero externo:

- El número de sucursales.
- Los siguientes valores por cada una de las sucursales:
  - Ingresos totales a repartir
  - Número de fabricantes
  - Parte que se lleva el diseñador
  - Retención fiscal del diseñador
  - Retención fiscal de los fabricantes

Por ejemplo, el siguiente fichero indica que hay dos sucursales. La primera tiene unos ingresos de 300 euros, 3 fabricantes, el diseñador se lleva 1/6, la retención del diseñador es del 20% y la de cada fabricante un 18%. Los datos para la segunda son 400 euros, 5 fabricantes, 1/4, 22% y 19%.

```
2
300 3 6 20 18
400 5 4 22 19
```

El programa tendrá que imprimir los salarios brutos y netos del diseñador y de los fabricantes por cada una de las sucursales, llamando a los métodos oportunos de la clase `Nomina`.

Finalidad: Diseño de una clase y trabajar con datos miembro constantes. Dificultad Media.

9. Defina una clase `Frase` para almacenar un conjunto de caracteres. Como dato miembro ha de usarse un objeto `vector` (de la STL) de `char`. 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.
- Sólo debe definir los datos miembro y métodos necesarios para resolver este ejercicio. No puede usarse el tipo `string`.
- 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 es `{'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.

Añadir 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.

10. Definir la clase `VectorParejasCaracterEntero` que permite almacenar un conjunto de parejas de la forma (carácter, entero). Cada pareja será un `struct` que llamaremos `ParejaCaracterEntero`, con un campo de tipo carácter y otro campo de tipo entero. Se pide crear un método de la clase `Frase` (Ejercicio 9) 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}`

    En la implementación de las clases utilizar la clase `vector`.

    Finalidad: Trabajar con vectores como datos miembro y pasados como parámetros. Dificultad Media.

11. (Bicicleta) Se quiere construir una clase para representar la tracción de una bicicleta, es decir, el conjunto de estrella (engranaje delantero), cadena y piñón (engranaje trasero). Supondremos que la estrella tiene tres posiciones (numeradas de 1 a 3, siendo 1 la estrella más pequeña) y el piñón siete (numeradas de 1 a 7, siendo 1 el piñón más grande). La posición inicial de marcha es estrella = 1 y piñón = 1.

    La clase debe proporcionar métodos para cambiar la estrella y el piñón, sabiendo que la estrella avanza o retrocede de 1 en 1 y los piñones cambian a saltos de uno o de dos. Si ha llegado al límite superior (inferior) y se llama al método para subir (bajar) la estrella, la posición de ésta no variará. Lo mismo se aplica al piñón.

    Cread un programa principal que lea desde un fichero externo los movimientos realizados e imprima la situación final de la estrella y piñón. Los datos se leerán en el siguiente formato: tipo de plato (piñón o estrella) seguido del tipo de movimiento. Para codificar esta información se usarán las siguientes letras: E indica una estrella, P un piñón, S para subir una posición, B para bajar una posición, T para subir dos posiciones y C para bajar dos posiciones. T y C sólo se aplicarán sobre los piñones.

    E S P S P S P S P C E S E B #

    En este ejemplo los movimientos serían: la estrella sube, el piñón sube en tres ocasiones sucesivas, el piñón baja dos posiciones de golpe, la estrella sube y vuelve a bajar. Supondremos

siempre que la posición inicial de la estrella es 1 y la del piñón 1. Así pues, la posición final será Estrella=1 y Piñón=2.

Mejorad la clase para que no permita cambiar la marcha (con la estrella o el piñón) cuando haya riesgo de que se rompa la cadena. Este riesgo se produce cuando la marcha a la que queremos cambiar es de la siguiente forma:

- Estrella igual a 1 y piñón mayor o igual que 5.
- Estrella igual a 2 y piñón o bien igual a 1 o bien igual a 7.
- Estrella igual a 3 y piñón menor o igual que 3.

Finalidad: Diseñar la interfaz de una clase. Dificultad Media.

12. (Empresa) Recuperad la solución del ejercicio 21 de la relación de ejercicios 2 (Empresa). Reescribid el programa principal usando una clase `Ventas` para gestionar los cómputos de las ventas realizadas. Únicamente se pide que se indiquen las cabeceras de los métodos públicos de la clase y las llamadas a éstos en el programa principal. No hay que implementar ninguno de los métodos. El programa principal sería de la siguiente forma:

```
cin >> identif_sucursal;
fichero_vacio = identif_sucursal == TERMINADOR;

if (!fichero_vacio){
   while (identif_sucursal != TERMINADOR){
      cin >> cod_producto;
      cin >> unidades_vendidas;

      --> Actualiza el número de unidades vendidas de la sucursal leida

      cin >> identif_sucursal;
    }

    --> Obtener el identificador y el número de ventas de la sucursal ganadora
}
```

Finalidad: Diseño de una clase. Dificultad Media.

13. (Ventas) Implementar los métodos de la clase `Ventas` del ejercicio anterior.

    Finalidad: Diseño de una clase. Dificultad Media.

14. (Examen) 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). Para representar una lista puede usar un vector de la STL o un array. 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 `GetEnunciado` que devuelva el enunciado de la pregunta i-ésima.
- Un método `GetRespuesta` 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.

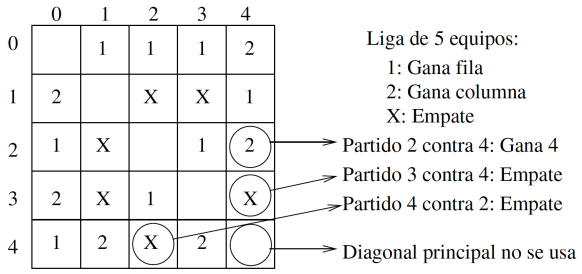
Una vez leído el examen se procederá a la evaluación de un número de alumnos dado desde la entrada estándar. Para ello el programa le mostrará las preguntas del examen a cada alumno y leerá sus respuestas. Al finalizar cada alumno la prueba, el programa le dirá su nota de acuerdo a los siguientes criterios:

- 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, así como implementar funciones externas a la misma.

Dificultad Baja.

15. Para gestionar un campeonato de n equipos se utiliza una matriz de tamaño $n \times n$. 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.

Implementar una clase `Liga` con al menos un método que devuelva un vector de enteros con los resultados finales de la liga. En este vector se 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.

Se pide lo siguiente:

- Cread dos versiones de la clase `Liga`:
  (a) Usando un `vector <vector <char>>` como dato miembro de la clase `Liga`.

(b) Usando una matriz clásica de doble corchete como dato miembro de la clase `Liga`

En ambas versiones, usad un `vector <int>` para devolver el vector de resultados finales.

- Cread también una clase `GeneradorLiga`, sin datos miembro y con un método que permita leer los datos de los resultados de los equipos desde un fichero (con redirección de la entrada) 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.

Finalidad: Trabajar con matrices. Dificultad Baja.

16. Se desea gestionar una librería. Para cada libro, únicamente se desea almacenar un identificador y la fecha de su publicación. Para ello, se pide definir las siguientes clases:

    - La clase `Identificador` contendrá un dato miembro privado de tipo `string` en el que se almacenará un identificador de 4 caracteres. Proporcionará, al menos, sendos métodos `EsIgual_a`, `EsMenor_que` para comparar con otro objeto de la clase `Identificador`. Diremos que un objeto identificador es menor que otro según lo indique el orden lexicográfico de sus datos miembros de tipo `string`. Por ejemplo, la cadena `ABZZ` es menor que `ACAA` y `AAAB` es menor que `AAAC`. Para comparar dos cadenas, usad el método `compare` de la clase `string`

      `cadena.compare(otra_cadena)`

      que devuelve 0 si son iguales, un entero negativo si `cadena` es menor que `otra_cadena` y un positivo en caso contrario.
    - La clase Fecha se construirá a partir del código proporcionado en las transparencias de clase. Hay que añadirle sendos métodos `EsIgual_a`, `EsMenor_que` para comparar con otro objeto de la clase `Fecha`. Diremos que un objeto de fecha es menor que otro según lo indique el orden natural de las fechas correspondientes.
    - La clase `Libro` contendrá como datos miembros un `Identificador` y una `Fecha`. Proporcionará, al menos, sendos métodos `EsIgual_a`, `EsMenor_que` para comparar con otro objeto de la clase `Libro`. Posteriormente se indica cuál es el criterio de comparación entre dos libros.
    - La clase `Biblioteca` contendrá un conjunto de libros. Puede usar como dato miembro privado un `vector` de la STL de `Libro`. Debe proporcionar al menos métodos para añadir y recuperar los datos de los libros.
    - La clase `LectorBiblioteca` para leer los datos desde un fichero y crear una biblioteca. Los datos estarán dispuestos de la siguiente forma:
      - En primer lugar aparece una cadena de caracteres con el identificador del libro. Este dato se leerá sobre un `string`, en la forma:
        `cin >> identificador;`
        La lectura de la cadena termina automáticamente cuando aparezca el primer espacio en blanco.
      - A continuación, separados por espacios en blanco, aparecen tres enteros con el día, mes y año de publicación.

Habrá tantas repeticiones como libros haya. El final del fichero vendrá marcado por la aparición de la cadena `FFFF`, como por ejemplo:

```
JFGT 30 1 2003 JGHT 26 2 1998 YFGT 30 1 2003 AUTQ 26 2 1998 FFFF
```

- La clase `ImpresorBiblioteca` para que imprima en pantalla todos los libros.

Se pide construir el método `Ordena` sobre la clase `Biblioteca` para que ordene los libros por orden de fecha de menor a mayor. A igualdad de fechas, ordenar por orden del identificador de menor a mayor. En el ejemplo anterior, la biblioteca quedaría como sigue:

```
AUTQ 26 2 1998 JGHT 26 2 1998 JFGT 30 1 2003 YFGT 30 1 2003
```

El algoritmo de ordenación es similar a cualquiera de los vistos en teoría. Simplemente cambia el tipo de dato del vector sobre los que se trabaja.

Cambiar ahora el método `EsMenor_que` de la clase `Libro` para que ordene por orden de fecha de mayor a menor. A igualdad de fechas, ordenar por orden del identificador de menor a mayor. En el ejemplo anterior, la biblioteca quedaría como sigue:

```
JFGT 30 1 2003 YFGT 30 1 2003 AUTQ 26 2 1998 JGHT 26 2 1998
```

Finalidad: Trabajar con un vector de objetos. Dificultad Media.

17. Sgeún un etsduio de una uivenrsdiad ignlsea, no ipmotra el odren en el que las ltears etsan ersciats, la úicna csoa ipormtnate es que la pmrirea y la útlima ltera etsén ecsritas en la psioción cocrrtea. El rsteo peuden estar ttaolmntee mal y aún pordás lerelo sin pobrleams. Etso es pquore no lemeos cada ltera por sí msima snio la paalbra cmoo un tdoo.

    Se pide crear la clase `Palabra` que permita almacenar un conjunto de caracteres que representarán una palabra. Definid un método `EsIgual` al que se le pase como parámetro otra palabra y determine si son iguales atendiendo al siguiente criterio: La primera letra de ambas palabras es igual, la última letra de ambas palabras también es igual y el resto de las letras son las mismas pero no están necesariamente en las mismas posiciones.