# Fundamentos de Programación.

# Guión de Prácticas.

Curso 2012/2013

# RELACIÓN DE PROBLEMAS IV. Vectores

**Importante: Para todos los ejercicios, se ha de diseñar una batería de pruebas.**
Si se va a leer desde un fichero, para poder leer carácter a carácter incluidos los espacios en blanco, debe realizar la lectura en la forma `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. En clase de teoría se ha visto la siguiente clase `MiVectorCaracteres`.

```
class  MiVectorCaracteres{
private:
   static const int TAMANIO = 50;
   char vector_privado[TAMANIO];
   int total_utilizados;
public:
   MiVectorCaracteres()
      :total_utilizados(0)
   {
   }
   int TotalUtilizados(){
      return total_utilizados;
   }
   void Aniade(char  nuevo){
      if (total_utilizados < TAMANIO){
         vector_privado[total_utilizados] = nuevo;
         total_utilizados++;
      }
   }
   char Elemento(int indice){
      return vector_privado[indice];
   }
};
```

   Añadir un método que nos diga 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.

   *Finalidad: Ejemplo de método que accede a las componentes del vector. Dificultad Baja.*

2. Sobre la clase `MiVectorCaracteres`, añadir un método con la cabecera

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

   para que modifique la componente con índice `indice_componente` y ponga en su lugar 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. Sobre la clase `MiVectorCaracteres`, añadir un método `IntercambiaComponentes` para intercambiar dos componentes del vector. Por ejemplo, si el vector contiene `{'h','o','l','a'}`, después de intercambiar las componentes 1 y 3, se quedaría con `{'h','a','l','o'}`.

   Añadid otro método para invertir el vector, de forma que si el vector contenía los caracteres `{'a','g','t','b','i','o'}`, después de llamar al método se quedará con `{'o','i','b','t','g','a'}`. Para implementar este método, llamad el método anterior `IntercambiaComponentes`.

   ¿Qué deberían hacer los métodos si los índices de componentes no son correctos?

   Imprimir las componentes del vector 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 del vector. Ya lo arreglaremos en el tema siguiente.

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

4. Sobre la clase `MiVectorCaracteres`, añadir un método `Elimina` para eliminar el carácter que se encuentre en una determinada posición, de forma que si el vector contenía los caracteres `{'h','o','l','a'}`, después de eliminar la componente con índice 2 (la tercera) se quedará con `{'h','o','a'}`.

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

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

5. Sobre la clase `MiVectorCaracteres`, se quiere añadir un 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'}`. Implementar el siguiente algoritmo:

   ```
   Recorrer todas las componentes del vector
      Si la componente es una mayúscula, borrarla
   ```

Para borrar la mayúscula, se desplazan una posición a la izquierda todas las componentes que hay a su derecha. Para ello, llamad al método `Elimina` que se ha definido en el ejercicio 4 de esta relación de problemas.

*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. 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.

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

## Problemas de Ampliación

7. Sobre la clase `MiVectorCaracteres`, añadir un método para eliminar el exceso de caracteres en blanco, es decir, que suprima todas las secuencias de espacios en blanco mayor de 1. Por ejemplo, si el vector original es `(' ','a','h',' ',' ',' ','c')`, el vector resultante debe ser `(' ','a','h',' ','c')`. 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.*

8. Sobre la clase `MiVectorCaracteres`, añadir un método `EliminaRepetidos` que quite los elementos repetidos, de forma que cada componente sólo aparezca una única vez. Por ejemplo, si el vector 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*) Usar un vector local `sin_repetidos` en el que almacenamos una única aparición de cada componente:

   ```
   Recorrer todas las componentes del vector original
      Si la componente NO está en el vector 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 del vector. Para borrar una componente, podríamos desplazar una posición a la izquierda, todas las componentes que estén a su derecha. El algoritmo quedaría:

```
Recorrer todas las componentes del vector 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 lo menor posible, e implementadla en el método `EliminaRepetidos`

*Dificultad Media.*

9. Supongamos un fichero en el que hay caracteres cualesquiera. Realizar un programa que vaya leyendo dichos caracteres hasta que se encuentre un punto `'.'`. Queremos contar el número de veces que aparece cada una de las letras mayúsculas (`'A'`, `...`, `'Z'`). Una posibilidad sería leer el carácter, y después de comprobar si es una mayúscula, ejecutar un conjunto de sentencias del tipo:

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

Sin embargo, este código es muy redundante. Proponed una solución e implementarla.

Para resolver este ejercicio puede usarse o bien un vector clásico de enteros, o bien un objeto de una clase que construyamos del tipo `MiVectorEnteros` (en cuyo caso, sustituiríamos los corchetes de acceso a las componentes por el correspondiente método)

*Finalidad: Diseñar un vector en el que los índices de las componentes representan algo*. *Dificultad Media.*

10. Sobre la clase `MiVectorCaracteres`, añadir un método `PosicionCaracteresConsecutivos` que calcule la posición donde se encuentre la primera secuencia de al menos $n$ caracteres consecutivos e iguales a uno que se pase como parámetro al método. Por ejemplo, en el vector de abajo, si $n = 3$, y el carácter buscado es `'a'`, entonces dicha posición es la 4.
`{'b','a','a','h','a','a','a','a','c','a','a','a','g'}`

*Dificultad Media.*

11. Escribir un programa que calcule, almacene en un vector y muestre por pantalla los $k$ primeros términos de la sucesión de Fibonacci de orden $n$. Tanto $n$ como $k$ serán enteros a leer desde el dispositivo de entrada por defecto.

    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:

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

    El programa debe definir una clase llamada `Fibonacci` para generar los números de Fibonacci y otra clase del tipo `MiVectorEnteros` para ir almacenándolos.

    *Dificultad Media.*

# Sesión 10

## ► Actividades a realizar en casa

### Vectores

***Actividad: Resolución de problemas.***

Resolved los ejercicios siguientes de la relación de problemas IV de Vectores.

- *Obligatorios:*

```
1 solucion1.cpp
2 solucion2.cpp
3 solucion3.cpp
4 solucion4.cpp
5 solucion5.cpp
```

- *Opcionales:*

```
6 solucion6.cpp
```