Problemas con vectores

1. Inversión de una secuencia[1]

Diseña un programa C++ que solicite en primer lugar al usuario un número positivo *n*, luego solicite *n* datos de tipo real y por último los escriba en la pantalla en orden inverso al introducido. Por ejemplo:

```
Introduzca un número positivo: 8
Introduzca 8 números reales: 5 9 -1 0 0.1 8 25 -25
La secuencia en orden inverso es
-25, 25, 8, 0.1, 0, -1, 9, 5
```

2. Contar en una secuencia[2]

Diseña un programa C++ que solicite en primer lugar al usuario un número positivo *n*, luego solicite *n* datos de tipo real y, a continuación, escriba en la pantalla el valor medio de los datos introducidos y cuántos de ellos son superiores a la media. Por ejemplo:

```
Introduzca un número positivo: 8
Introduzca 8 números reales: 5  9.5  -1.5  0  -100  8  25  -25
La media de los datos introducidos es -9.875 y 6 de ellos son superiores a la media.
```

Nota: Si has resuelto los problemas 1º y 2º escribiendo solo una función `main` en para cada uno de ellos, habrás repetido el código que lee los datos del vector desde el teclado. Lo más adecuado hubiera sido utilizar una función que se reutilizara en los dos problemas. Las soluciones publicadas resuelven el problema tanto sin utilizar funciones (ficheros «1a-inversion-secuencia-sin-funciones» y «2a-superiores-media-sin-funciones.cpp») tanto como con funciones («1a-inversion-secuencia-main» y «2b-superiores-media-main.cpp»). En la versión con funciones, como la función que lee el vector del teclado podría reutilizarse en ambos, ha sido definida en un módulo denominado «leer-vector-reales».

Comprenderás mejor las soluciones con funciones cuando hayamos visto en clase de teoría la utilización de vectores como parámetros de funciones (martes 2 en el grupo 412 y viernes 6 en el grupo 411).

3a. Contar en un vector

Escribe un programa que declare un vector de 21 componentes de tipo entero y lo inicalice como

```
{1, -5, -8, 6, 8, -3, -4, -6, 6, -1, 5, -8, 1, 9, 5, -9, 5, 8, 9, -2, -9}
```

El programa debe calcular y escribir en pantalla cuantos datos de ese vector son estrictamente positivos:

```
El vector tiene 11 datos estrictamente positivos.
```

[1] Basado en: Alberto Ciriano y Luis A. Gambau. *Enunciados de problemas de la asignatura Programación. Electrónica Industrial. Curso 2001/2002.* Problema 6.1

[2] Basado en: Alberto Ciriano y Luis A. Gambau. *Problemas resueltos de la asignatura Programación. Electrónica Industrial. Curso 2001/2002.* Problema 6.1

Funciones con vectores

Los siguientes problemas se centran en trabajar con vectores y piden solo el cuerpo de funciones que tienen vectores como parámetros. Podrás hacerlos mejor cuando hayamos visto en clase de teoría la utilización de vectores como parámetros de funciones (martes 2 en el grupo 412 y viernes 6 en el grupo 411).

3b. Contar en un vector

Diseña la siguiente función:

```
/*
 * Pre:  El vector «v» tiene al menos «n» componentes.
 * Post: Ha devuelto el número de datos positivos almacenados en las primeras «n»
 *       componentes del vector «v».
 */
unsigned int numPositivos(const int v[], const unsigned int n);
```

Nota: La solución al problema 3a podría utilizar la función `numPositivos` que se pide en este problema 3b.

4. Copiar un vector

En C++, el operador de asignación no está disponible para vectores. Diseña la siguiente función, que permitiría, cuando sea invocada, copiar todos los datos de un vector en otro:

```
/*
 * Pre:  Los vectores «original» y «copia» tienen al menos «n» componentes cada uno.
 * Post: Ha copiado las primeras «n» componentes del vector «original» en las
 *       primeras «n» componentes del vector «copia».
 */
void copiar(const int original[], const unsigned int n, int copia[]);
```

5. Comparar dos vectores

En C++, el operador de comparación no está disponible para vectores. Diseña la siguiente función, que permite, cuando es invocada, averiguar si los datos de dos vectores, componente a componente, son iguales:

```
/*
 * Pre:  Los vectores «v» y «w» tienen al menos «n» componentes cada uno.
 * Post: Ha devuelto ture si y solo si los datos de las primeras «n» componentes de
 *       los vectores «v» y «w» son iguales.
 */
bool sonIguales(const int v[], const int w[], const unsigned int n);
```

6. Cálculo de la moda

En términos estadísticos, la *moda* de una distribución de datos es el valor que aparece con una mayor frecuencia, es decir, el que más se repite.

Dadas las siguientes colecciones de datos de tipo entero, indica cuál es la moda de cada una de ellas:

{4, 7, 5, 5, 1, 7, 6, 9, 2, 2, 5, 0, 7, 9, 5, 6, 5, 0, 8, 0, 2}:
{1, 5, 8, 6, 8, 3, 4, 6, 6, 1, 5, 8, 1, 9, 5, 9, 5, 8, 9, 2, 9}:

Diseña el código de las siguientes funciones:

```
/*
 * Pre:  El vector «v» tiene «n» componentes y se cumple que 0 ≤ i < n.
 * Post: Ha devuelto el número de veces que está repetido el dato «i»-ésimo
 *       del vector «v».
 */
int numeroRepeticiones(const int v[], const unsigned int i, const unsigned int n);
```

```
/*
 * Pre:  El vector «v» tiene «n» componentes.
 * Post: Ha devuelto la moda de los datos almacenados en el vector «v».
 */
int moda(const int v[], const unsigned int n);
```

7. Búsqueda de un dato en un vector

```
/*
 * Pre:  El vector «datos» tiene «n» componentes.
 * Post: Ha devuelto el valor true si y solo si un dato igual al valor del parámetro
 *       «buscado» está almacenado en el vector «datos».
 */
bool esta(const int datos[], const int n, const unsigned int buscado);
```

8. Orden de los datos

Diseña la siguiente función:

```
/*
 * Pre:  El vector «v» tiene «n» componentes.
 * Post: Ha devuelto true si y solo si los datos del vector «v» están ordenados
 *       de forma no decreciente.
 */
bool estaOrdenada(const int v[], const unsigned int n);
```

9. Filtrar los datos en un vector

Diseña la siguiente función:

```
/*
 * Pre:  El vector «v» tiene al menos «n» componentes; sea Np el número de
 *       datos positivos del vector «v»: el vector «p» tiene al menos Np componentes.
 *       El valor del parámetro «numPositivos» no está definido.
 * Post: Las primeras Np componentes del vector «p» almacenan los Np datos positivos
 *       de las primeras «n» componentes del vector «T» y numPositivos = Np.
 */
void copiaPositivos (const int v[], const unsigned int n,
                     int p[], unsigned int & numPositivos);
```

10. Secuencia de Kolakoski

Basado en el examen del 12 de septiembre de 2011 (2,5 puntos)

La secuencia de Kolakoski es una secuencia infinita compuesta por bloques integrados por unos o por doses. Un bloque puede estar compuesto por un único dígito (1 o 2) o dos dígitos iguales (11 o 22). El dígito utilizado en un bloque es distinto al dígito utilizado en los bloques inmediatamente anterior y posterior.
Cada uno de los dígitos de la secuencia informa de la longitud de uno de los bloques posteriores. Así, el *n*-ésimo dígito de la secuencia (que puede ser un uno o un dos) informa de la longitud del (*n*+1)-ésimo bloque de la secuencia.
La secuencia se construye del siguiente modo:

- Por definición, el primer bloque de la secuencia tiene longitud 1 y está formado por el dígito **1.**
- Este 1 indica que el siguiente bloque (el segundo de la secuencia) tiene longitud 1. Como hay que alternar unos y doses, el siguiente bloque está formado por el dígito 2. Los dos primeros bloques definen la subsecuencia **12.**
- El segundo dígito, que es un 2, indica que el tercer bloque tiene longitud 2. Como hay que alternar unos y doses, el tercer bloque está formado por los dígitos 11. Los tres primeros bloques de la secuencia totalizan cuatro dígitos: **1211.**
- El tercer dígito, un 1, indica que el cuarto bloque tiene longitud 1. Por lo tanto, los cuatro primeros bloques de la secuencia son **12112.**
- El cuarto dígito, que es un 1, indica que el quinto bloque tiene longitud 1. Los cinco primeros bloques de la secuencia están formados por los dígitos **121121.**
- Sucesivamente y de este mismo modo se siguen añadiendo bloques a la secuencia. Por ejemplo, los 47 primeros bloques de la secuencia están formados por los 70 dígitos siguientes: **12112122122112112212112122112112122122112122112112212211212212211211.**

Se pide completar la siguiente función, añadiendo el código C++ correspondiente:

```
/* Pre:  El vector «secuencia» tiene «longitud» componentes.
 * Post: Ha escrito en el vector «secuencia» los primeros «longitud» dígitos de la
 *       secuencia de Kolakoski.
 */
void kolakoski(unsigned int secuencia[], const unsigned int longitud);
```