



Problemas con vectores

Los siguientes problemas se pueden resolver a través de una única función main, sin hacer uso de funciones ni procedimientos auxiliares, aunque nada impide utilizarlos.

1. Contar en un vector

Escribe un programa que declare un vector de 21 componentes de tipo entero y lo inicialice 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 contar y escribir en pantalla cuántos datos de ese vector son estrictamente positivos:

El vector tiene 11 datos estrictamente positivos.

2. Inversión de una secuencia¹

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

3. Contar en una secuencia²

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 2º y 3º escribiendo solo una función main para cada uno de ellos, habrás repetido el código que lee los datos del vector desde el teclado. Lo más adecuado para esto hubiera sido utilizar un procedimiento que se reutilizara en los dos problemas. Las soluciones publicadas resuelven el problema sin utilizar funciones auxiliares (ficheros «2a-inversion-secuencia-sin-funciones» y «3a-superiores-media-sin-funciones.cpp») y también utilizando el procedimiento auxiliar mencionado («2b-inversion-secuencia-main» y «3b-superiores-media-main.cpp»). En la versión con procedimiento auxiliar, como el procedimiento que lee el vector del teclado puede usarse en la solución de ambos problemas, ha sido definido en un módulo denominado «leer-vector-reales».

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 6.2



Funciones con vectores

Los siguientes problemas se centran en trabajar con vectores y piden solo el cuerpo de funciones y procedimientos que tienen vectores como parámetros.

4. Contar en un vector

Diseña la siguiente función:

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

Nota: La solución al problema 1 podría utilizar la función numPositivos que se pide en este problema 4.

5. Copiar un vector

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

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

6. 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 de enteros son iguales componente a componente:

```
/*  
 * Pre: Los vectores «v» y «w» tienen al menos «n» componentes cada uno.  
 * Post: Devuelve true 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 n);
```



7. 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.

Antes de programar nada, 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 dos funciones:

```
/*  
 * Pre: EL vector «v» tiene «n» componentes y se cumple que  $0 \leq i < n$ .  
 * Post: Devuelve el número de veces que está repetido el dato  $v[i]$  en el vector «v».  
 */  
unsigned numeroRepeticiones(const int v[], const unsigned i, const unsigned n);
```

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

8. Búsqueda de un dato en un vector

```
/*  
 * Pre: EL vector «datos» tiene «n» componentes.  
 * Post: Devuelve 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 unsigned n, const int buscado);
```

9. Orden de los datos

Diseña la siguiente función:

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

10. Filtrar los datos en un vector

Diseña la siguiente función:

```

/*
 * Pre: Los vectores «v» y «p» tienen al menos «n» componentes cada uno; el valor
 * del parámetro «numPositivos» no está definido.
 * Post: Tras ejecutar esta función, «numPositivos» es igual al número de datos positivos
 * que hay en el vector «v» y las primeras «numPositivos» componentes del vector «p»
 * almacenan los datos positivos de las primeras «n» componentes del vector «T».
 *
 * Por ejemplo, si se invoca a esta función en el siguiente entorno:
 *     int vector[9] = {3, 0, 5, -1, 2, -6, -4, 8, -9};
 *     int positivos[9];
 *     unsigned int numPos;
 *     copiaPositivos(vector, 9, positivos, numPos);
 * tras ejecutar la función, numPos = 5 y el vector «positivos» almacenaría en
 * sus primeras 5 componentes los datos {3, 0, 5, 2, 8}.
 */
void copiaPositivos(const int v[], const unsigned n,
                    int p[], unsigned &numPositivos);

```

11. 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:
1211212212211211221211212211211212212211212212112112212211212212211211.

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

```

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

```