Recorridos de vectores de datos de tipos simples

Estos problemas ya se publicaron en el tema de vectores (https://miguel-latre.github.io/transparencias/problemas-7-problemas-con-vectores.pdf).

1. 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);
```

2. 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 «v».
 *
 *       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 int n,
                    int p[], unsigned int& numPositivos);
```

3. 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 y «n» > 0.
 * Post: Ha devuelto la moda de los datos almacenados en el vector «v».
 */
int moda(const int v[], const unsigned int n);
```

Recorridos de vectores de cadenas

Estos problemas ya se publicaron en el tema de vectores (https://miguel-latre.github.io/transparencias/problemas-7-problemas-con-vectores.pdf).

1. Recorrido de una cadena de caracteres

```
/*
 * Pre:  ---
 * Post: Ha devuelto el número de caracteres de la cadena «cad» cuyo valor es el de
 *       una letra mayúscula o minúscula del alfabeto inglés.
 */
unsigned int contarLetras(const string cad);
```

2. *CamelCase*

Se denomina *CamelCase* al estilo de escritura que se aplica a frases o palabras compuestas, cuando las palabras simples que las componen se escriben todas juntas unas a otras, sin dejar espacios intermedios, y los inicios de las distintas palabras se marcan intercalando letras mayúsculas, con la posible excepción de la letra inicial de la primera palabra. Por ejemplo, el apellido escocés *MacLean*, la marca *CinemaScope*, la fórmula química *NaCl* o el identificador de C++ *numeroPalabrasEnCamelCase* son ejemplos en los que se ha aplicado el estilo *CamelCase*.

Diseña la siguiente función C++:

```
/*
 * Pre:  Todos los caracteres de «cadena» son letras del alfabeto inglés.
 * Post: Ha devuelto el número de palabras individuales que forman «cadena».
 */
unsigned int numeroPalabrasEnCamelCase(string cadena);
```

A modo de ejemplo, las siguientes invocaciones a `numeroPalabrasEnCamelCase` deberían producir los siguientes resultados:

`numeroPalabrasEnCamelCase("")`	debe devolver 0
`numeroPalabrasEnCamelCase("a")`	debe devolver 1
`numeroPalabrasEnCamelCase("EINA")`	debe devolver 1
`numeroPalabrasEnCamelCase("camelCase")`	debe devolver 2
`numeroPalabrasEnCamelCase("iPad")`	debe devolver 2
`numeroPalabrasEnCamelCase("ArrayIndexOutOfBoundsException")`	debe devolver 6
`numeroPalabrasEnCamelCase("numeroPalabrasEnCamelCase")`	debe devolver 5

Recorridos de vectores de registros

Se va a trabajar con datos de tipo `Permiso`, definido en el módulo `permiso` correspondiente a la clase de problemas de la semana pasada. Como referencia, se reproducen a continuación las declaraciones de su fichero de interfaz (las declaraciones junto con sus especificaciones pueden consultarse en el enunciado de la semana pasada, y su implementación está publicada en el repositorio GitHub de la asignatura):

```
const int MAX_NUM_MOVIMIENTOS = 200;
const int MESES_NOVEL = 12;

struct Permiso {
    string nombreCompleto;
    unsigned int antiguedadMeses;
    int movimientos[MAX_NUM_MOVIMIENTOS];
    unsigned int numMovimientos;
};

void inicializarComoNuevo(Permiso& p, const string nombre);
bool esNovel(const Permiso& p);
int puntos(const Permiso& p);
void registrarSancion(Permiso& p, const unsigned int sancion);
void registrarBonificacion(Permiso& p, const unsigned int bonificacion);
```

Completa el código de las siguientes funciones que trabajan con vectores de registros del tipo `Permiso`:

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha devuelto el número de permisos de conducir de las primeras «n»
 *       componentes del vector «v» con una cantidad de puntos negativa o igual a 0.
 */
unsigned int contarSinPuntos(const Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes y «n» > 0.
 * Post: Ha devuelto el permiso de conducir de entre las primeras «n» componentes del
 *       vector «v» que tiene el menor saldo de puntos.
 */
Permiso peorConductor(const Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha devuelto un índice de una componente de entre las primeras «n»
 *       componentes del vector «v» que contiene un permiso con un «puntosBuscados»
 *       puntos, o un valor negativo si no existe ninguno en el vector.
 */
int buscarPorPuntos(const Permiso v[], const unsigned int n,
                    const int puntosBuscados);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha recorrido las primeras «n» componentes el vector «v» y ha aumentado la
 *       antigüedad de todos los permisos en un mes.
 */
void actualizarMes(Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha recorrido las primeras «n» componentes el vector «v» y, cuando ha
 *       encontrado permisos en ellas correspondientes a conductores que han dejado
 *       de ser noveles (conductores con exactamente 12 meses de antigüedad), les ha
 *       bonificado con 4 puntos.
 *       Ha devuelto el número de permisos de conductores a los que se ha
 *       bonificado por dejar de ser noveles.
 */
int bonificarPorDejarDeSerNovel(Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  Los vectores «v» y «resultado» tienen al menos «nV» componentes cada uno.
 * Post: El vector «resultado» contiene, en sus primeras «nR»
 *       componentes, únicamente aquellos permisos de las primeras «nV» componentes
 *       del vector «v» que tienen un saldo de puntos estrictamente positivo.
 */
void purgar(const Permiso v[], const unsigned int nV,
            Permiso resultado[], unsigned int& nR);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha devuelto «true» si y solo si las primeras «n» componentes del vector «v»
 *       están ordenadas de forma que los permisos de sus componentes tienen
 *       valores de puntos no decrecientes.
 */
bool estaOrdenadaPorPuntos(const Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Ha devuelto true si y solo si las primeras «n» componentes del vector «v»
 *       están clasificadas de forma tal que todos los permisos correspondientes a
 *       conductores noveles aparecen primero (en las componentes de índices más
 *       bajos) y todos los correspondientes a conductores experimentados, después
 *       (en las componentes de índices más altos).
 */
bool estaOrdenadaPorNovel(const Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Las primeras «n» componentes del vector «v» son una permutación de los
 *       permisos que había inicialmente en esas mismas primeras «n» componentes del
 *       vector «v» y están clasificadas de forma que todos los permisos
 *       correspondientes a conductores noveles aparecen primero (en las
 *       componentes de índices más bajos) y todos los correspondientes a
 *       conductores experimentados, después (en las componentes de índices más
 *       altos).
 */
void clasificarPorNovel(Permiso v[], const unsigned int n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Las primeras «n» componentes del vector «v» son una permutación de los
 *       permisos que había inicialmente en esas mismas primeras «n» componentes del
 *       vector «v» y están ordenadas de forma que tienen valores de puntos no
 *       decrecientes.
 */
void ordenarPorPuntos(Permiso v[], const unsigned int n);
```