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 completas 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 unsigned MAX_NUM_MOVIMIENTOS = 200;
const unsigned MESES_NOVEL = 12;

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

void inicializarComoNuevo(Permiso &p, const string nombre);
bool esNovel(const Permiso &p);
int puntos(const Permiso &p);
void registrarSancion(Permiso &p, const unsigned sancion);
void registrarBonificacion(Permiso &p, const unsigned 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: Devuelve 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 contarSinPuntos(const Permiso v[], const unsigned n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes y «n» > 0.
 * Post: Devuelve 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 n);
```

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

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Recorre las primeras «n» componentes del vector «v», aumentando la antigüedad de todos
 *       los permisos en un mes.
 */
void actualizarMes(Permiso v[], const unsigned n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Recorre las primeras «n» componentes del vector «v» y, cuando encuentra permisos en
 *       ellas correspondientes a conductores que han dejado de ser noveles (conductores con
 *       exactamente 12 meses de antigüedad), les bonifica con 4 puntos. Devuelve el número de
 *       permisos de conductores a los que se bonifica por dejar de ser noveles.
 */
unsigned bonificarPorDejarDeSerNovel(Permiso v[], const unsigned 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 nV,
            Permiso resultado[], unsigned &nR);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Devuelve «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 estaOrdenadoPorPuntos(const Permiso v[], const unsigned n);
```

```
/*
 * Pre:  «v» tiene al menos «n» componentes.
 * Post: Devuelve «true» si y solo si las primeras «n» componentes del vector «v»
 *       están distribuidas 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 estaDistribuidoPorNovel(const Permiso v[], const unsigned 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 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 n);
```