



1. Representación parcial de permisos de conducción

Los problemas planteados en esta clase van a utilizar registros de un tipo denominado Permiso, que representan información relativa a permisos de conducir. En este conjunto de problemas planteados, vamos a considerar **únicamente** la siguiente información asociada a los permisos de conducir:

- El nombre completo del conductor o conductora.
- Su antigüedad, expresada como una cantidad entera de meses completos contados desde el día de su expedición.
- El historial de movimientos del número de puntos asociados al permiso: se desea tener constancia del número inicial de puntos asociados al permiso cuando este fue expedido y de las cuantías en puntos de las sucesivas bonificaciones y sanciones que la persona propietaria del mismo haya tenido.

Por ejemplo, consideremos una persona a la que le expidieron su permiso como novel en enero del año 2000 con 8 puntos. En 2002 fue bonificado con 4 puntos; en 2005, con 2 puntos más. En enero 2008 fue bonificado de nuevo con 1 punto y, 7 meses después, sancionado con 6 puntos. En el historial de movimientos del dato de tipo Permiso de esta persona, estaríamos interesados en almacenar los datos enteros {8, 4, 2, 1, -6}.

De momento, no se desea almacenar información sobre las fechas en las que se produjeron las bonificaciones o las sanciones, aunque no se descarta para un futuro.

Se estima que ningún conductor va a tener más de 200 movimientos en su historial de puntos durante toda la vigencia de su permiso de conducción.

Definición del tipo

Sin mirar la solución que aparece más abajo, define el tipo denominado Permiso para que las variables de dicho tipo puedan reflejar la información previamente comentada.

Implementación de un módulo denominado permiso

A continuación, se muestra el fichero de cabecera (permiso.h) de un módulo denominado permiso, que incluye la definición del tipo Permiso y algunas funciones para trabajar con registros de tipo Permiso. Se pide el código del fichero de implementación (permiso.cpp) del módulo permiso.

```
/* Máxima longitud de un nombre completo. */
const int MAX_LONG_NOMBRE = 200;

/* Estimación del máximo número de movimientos del historial de puntos. */
const int MAX_NUM_MOVIMIENTOS = 200;

/*
 * Los registros de tipo Permiso representan (de forma muy parcial) permisos de conducir por
 * puntos. Únicamente tenemos en cuenta el nombre del conductor, su antigüedad en meses y el
 * historial de puntos (asignación inicial, bonificaciones y sanciones).
 */
struct Permiso {
    char nombreCompleto[MAX_LONG_NOMBRE];
    int antigüedadMeses;
    int movimientosPuntos[MAX_NUM_MOVIMIENTOS];
    int numMovimientos;
    // Aquí iría la definición de otros campos DNI, fecha de expedición, tipo de carnet, ...
};

/*
 * Pre: La cadena «nombre» no tiene más de MAX_LONG_NOMBRE caracteres.
 * Post: Ha inicializado el permiso «conductorNovel» de forma que representa el
 * permiso de conducir de una persona llamada «nombre» que acaba de
 * obtenerlo.
 */
void inicializarComoNovel(Permiso& conductorNovel, const char nombre[]);
```



```
/*
 * Pre: ---
 * Post: Ha devuelto «true» si y solo si el titular del permiso «p» es un conductor novel.
 */
bool esNovel(const Permiso& p);

/*
 * Pre: ---
 * Post: Ha devuelto la cantidad de puntos asociados al permiso de conducir «p».
 */
int puntos(const Permiso& p);

/*
 * Pre: 0 < sancion <= 6
 * Post: Ha registrado entre Los movimientos de puntos del permiso «p» una
 *       sanción de «sancion» puntos.
 */
void registrarSancion(Permiso& p, const int sancion);

/*
 * Pre: puntos > 0
 * Post: Ha registrado entre Los movimientos de puntos del permiso «p» una
 *       bonificación de «bonificacion» puntos o inferior, en el caso de que se
 *       sobrepasara la cantidad legal máxima de 15 puntos.
 */
void registrarBonificacion(Permiso& p, const int bonificacion);
```

2. Representación canónica de números racionales

Un número racional se representa como un cociente de dos enteros, denominados numerador y denominador. El denominador no puede ser igual a 0. Ejemplos de números racionales:

$$\frac{1}{3}, \frac{7}{5}, \frac{-2}{17}, \frac{4}{-9}, \frac{-2}{-3}$$

Dado un número racional, hay un número infinito de números racionales equivalentes a él:

$$\frac{-1}{3} = \frac{1}{-3} = \frac{-2}{6} = \frac{2}{-6} = \frac{-3}{9} = \frac{3}{-9} = \frac{-4}{12} = \frac{4}{-12} = \dots$$

Se denomina *representante canónico* de un conjunto de números racionales equivalentes a aquel cuyo denominador es positivo y su numerador y denominador son primos entre sí. En el ejemplo anterior, el representante canónico del conjunto es el racional $\frac{-1}{3}$.

Dado un número racional, interesará sustituirlo por un racional equivalente que sea el representante canónico del conjunto de racionales equivalentes al primero. Esta operación la denominaremos *reducción del racional*, obteniendo como resultado su representante canónico.

Dado el siguiente fichero de interfaz (racional.h) de un módulo denominado racional, se debe:

- Definir un tipo denominado Racional, que permita representar números racionales canónicos.
- Completar el código de las funciones del módulo racional, en su correspondiente fichero de implementación racional.cpp.
- Añadir un módulo principal con una función main que, para comprobar el funcionamiento del módulo, escriba en la pantalla el racional $\frac{9}{8}$ resultante de realizar la siguiente operación: $\left[\left(\frac{2}{4} + \frac{1}{6}\right) - \frac{8}{3}\right] \times \left(\frac{6}{8} \div \frac{4}{-3}\right)$.

```
/*
 * Representación de números racionales canónicos.
 */
struct Racional {
    ...
};
```



Programación 1

Problemas con registros

```
/*
 * Pre: denominador  $\neq 0$ 
 * Post: Ha devuelto un registro de tipo Racional cuyo valor es el representante canónico de la
 *       fracción numerador/denominador.
 */
Racional definirRacional(const int numerador, const int denominador);

/*
 * Pre: «a» y «b» son racionales representantes canónicos.
 * Post: Ha devuelto  $a + b$ .
 */
Racional sumar(const Racional a, const Racional b);

/*
 * Pre: «a» y «b» son racionales representantes canónicos.
 * Post: Ha devuelto  $a - b$ .
 */
Racional restar(const Racional a, const Racional b);

/*
 * Pre: ---
 * Post: Ha devuelto  $-a$ .
 */
Racional opuesto(const Racional a);

/*
 * Pre: «a» y «b» son racionales representantes canónicos.
 * Post: Ha devuelto  $a \times b$ .
 */
Racional multiplicar(const Racional a, const Racional b);

/*
 * Pre: «a» y «b» son racionales representantes canónicos y  $b \neq 0$ .
 * Post: Ha devuelto  $a \div b$ .
 */
Racional dividir(const Racional a, const Racional b);

/*
 * Pre:  $a \neq 0$ 
 * Post: Ha devuelto  $1/a$ 
 */
Racional inverso(const Racional a);

/*
 * Pre: ---
 * Post: Ha devuelto el valor real de «a»
 */
double valorReal(const Racional a);

/*
 * Pre: ---
 * Post: Ha escrito el racional «a» en la pantalla.
 */
void escribir(const Racional a);

/*
 * Pre: ---
 * Post: Ha devuelto true si y solo si los racionales «a» y «b» son iguales.
 */
bool sonIguales(const Racional a, const Racional b);
```