Módulo de biblioteca «venta»

Los problemas planteados en esta clase van a utilizar vectores y registros de un tipo denominado «Venta», que representan información relativa a las distintas ventas de productos que realiza una empresa. Por cada venta, la empresa registra información acerca del código del producto vendido, el código del cliente a quien se ha vendido el producto, la cantidad de producto vendido y el precio unitario al que se ha vendido.

A continuación, se muestra la definición del tipo Venta con la que se va a trabajar.

```
/*
 * Los registros de tipo «Venta» gestionan la información asociada a ventas
 * realizadas por una empresa, según se especifica en el enunciado.
 */
struct Venta {
    unsigned producto;       // Código del producto vendido
    unsigned cliente;        // Código del cliente a quien se ha vendido
    int cantidad;            // Cantidad de producto que se ha vendido
    double precioUnitario;   // Precio unitario al que se ha vendido el producto
};
```

Ficheros de texto que almacenan ventas

Disponemos de ficheros de texto que almacenan los datos de cada una de las ventas realizadas por la empresa referida anteriormente. La estructura del fichero responde a la siguiente sintaxis:

```
<fichero-ventas> ::= { <venta> <fin-de-línea> }
<venta> ::= <producto> <separador> <cliente> <separador> <cantidad> <separador>
            <precio-unitario>
<producto> ::= literal-entero
<cliente> ::= literal-entero
<cantidad> ::= literal-entero
<precio-unitario> ::= literal-real
<separador> ::= ( " " | "\t" ) { " " | "\t" }
<fin-de-línea> ::= "\n"
```

Por ejemplo, un fichero que respondiera a dicha sintaxis podría tener el siguiente contenido:

```
117 120552 120 3.15
122 130922  65 6.40
105 120552 100 3.16
154 137054  75 0.98
```

El contenido del ejemplo anterior representa un total de 4 ventas. La primera línea representa una venta de 120 unidades del producto de código 117 al cliente de código 120552, a un precio unitario de 3.15 €. La segunda línea representa una venta de 65 unidades del producto de código 122 al cliente 130922 a un precio de 6.40€ cada una, y así sucesivamente.

Problema 1.º

Diseñad la siguiente función, que tiene como objetivo facilitar, en los programas que siguen, la lectura de una venta completa de un flujo asociado a un fichero que cumple con la sintaxis de ficheros de ventas. La función recibe como parámetro de entrada un flujo de lectura ya asociado a un fichero y del que se han podido leer ya algunas líneas completas. La función lee la siguiente línea pendiente de leer y asigna a los campos del registro venta los datos que ha leído. Al acabar, el flujo queda en un estado tal que podría volverse a invocar esta misma función para leer los datos de la siguiente venta del flujo.

```
/*
 * Pre:  El flujo «f» está asociado con un fichero de texto que cumple con la
 *       sintaxis de los ficheros de ventas establecida en el enunciado y en
 *       disposición de leer desde el principio de una línea.
 * Post: Intenta leer la línea mencionada en la precondición y, si no se terminan los
 *       datos del fichero en dicho intento, almacena en los campos del parámetro
 *       «venta» el código del producto vendido leído del fichero, el código del
 *       cliente a quien se ha vendido, la cantidad de producto y el precio unitario
 *       que se ha vendido. Devuelve «true» si los datos del fichero no se han
 *       terminado y, por lo tanto, se han podido leer los datos mencionados.
 *       Devuelve «false» en caso contrario.
 */
bool leerSiguienteVenta(istream &f, Venta &venta);
```

En los siguientes problemas, se puede hacer uso de esta función para simplificar la lectura de los datos del fichero.

Problema 2.º

Diseñad la función `totalFactura` que se especifica a continuación. No es necesario utilizar ningún vector adicional para su resolución.

```
/*
 * Pre:  Existe un fichero de ventas con el nombre «nombreFichero» accesible
 *       para su lectura y que cumple con la sintaxis de la regla <fichero-ventas>
 *       establecida en el enunciado.
 * Post: Devuelve la cantidad total a facturar al cliente cuyo código es igual a
 *       «clienteFactura» por las ventas que le corresponden registradas en
 *       el fichero de ventas de nombre «nombreFichero». Si el fichero
 *       «nombreFichero» no se puede abrir, devuelve -1.
 */
double totalFactura(const string nombreFichero, const unsigned clienteFactura);
```

Problema 3.º

Diseñad la función `eliminarErroneos` que se especifica a continuación. No es necesario utilizar ningún vector adicional para su resolución.

```
/*
 * Pre:  Existe un fichero de ventas con el nombre «nombreFicheroOriginal»
 *       accesible para su lectura y que cumple con la sintaxis de la regla
 *       <fichero-ventas> y es posible crear o reescribir el
 *       fichero de nombre «nombreFicheroFinal» para su escritura.
 * Post: Copia en el fichero de nombre «nombreFicheroFinal» las ventas almacenadas en
 *       el fichero de nombre «nombreFicheroOriginal» que no son erróneas y solo
 *       esas. Una venta es considerada errónea cuando la cantidad o el precio
 *       unitario son NO son positivos.
 */
void eliminarErroneos(const string nombreFicheroOriginal,
                      const string nombreFicheroFinal);
```

Problema 4.º

```
/*
 * Pre:  Existe un fichero de ventas con el nombre «nombreFichero» accesible
 *       para su lectura y que cumple con la sintaxis de la regla <fichero-ventas>.
 * Post: Devuelve el número de clientes diferentes cuyas ventas están
 *       registradas en el fichero de ventas de nombre «nombreFichero». Si el fichero
 *       no se puede abrir, devuelve -1.
 */
int numClientesDistintos(const string nombreFichero);
```

Problema 5.º

Diseñad la función `leerVentas` que se especifica a continuación:

```
/*
 * Pre:  Existe un fichero de ventas con el nombre «nombreFichero» accesible
 *       para su lectura y que cumple con la sintaxis de la regla <fichero-ventas> y
 *       el número de ventas almacenados en el mismo es
 *       menor o igual a la dimensión del vector «ventas».
 * Post: Si el fichero «nombreFichero» se ha podido abrir, asigna a «nVentas» el
 *       número de ventas del fichero, almacena las primeras «nVentas» componentes del
 *       vector «ventas» la información de las ventas almacenadas en el fichero y
 *       asigna a «lecturaOk» el valor «true». En caso contrario, asigna a
 *       «lecturaOk» el valor «false».
 */
void leerVentas(const string nombreFichero,
                Venta ventas[], unsigned &nVentas, bool &lecturaOk);
```

Problema 6.º

Diseñad la función `guardarVentas` que se especifica a continuación:

```
/* Pre:  ---
 * Post: Creado un fichero de nombre «nombreFichero» en el que ha almacenado la
 *       información de las «n» primeras componentes del vector «ventas» siguiendo la
 *       sintaxis de los ficheros de ventas establecida en el enunciado. Si el
 *       fichero se ha podido crear sin problemas, ha asignado a «escrituraOk» el
 *       valor «true». En caso contrario, ha asignado a «escrituraOk» el valor
 *       «false».
 */
void guardarVentas(const string nombreFichero,
                   const Venta ventas[], const unsigned n, bool &escrituraOk);
```