VNIVERSITAT DE VALÈNCIA

Laboratorio de PROGRAMACIÓN
Grado en Ingeniería Multimedia (1º)
Curso 2021-22

Práctica Nº 7: Aplicación de Colas

Periodo de realización: Semana del **9 al 13/05/2020**

Objetivos

- Utilizar el tipo de datos Cola para resolver un problema.
- Implementar una aplicación informática a partir de sus requisitos.

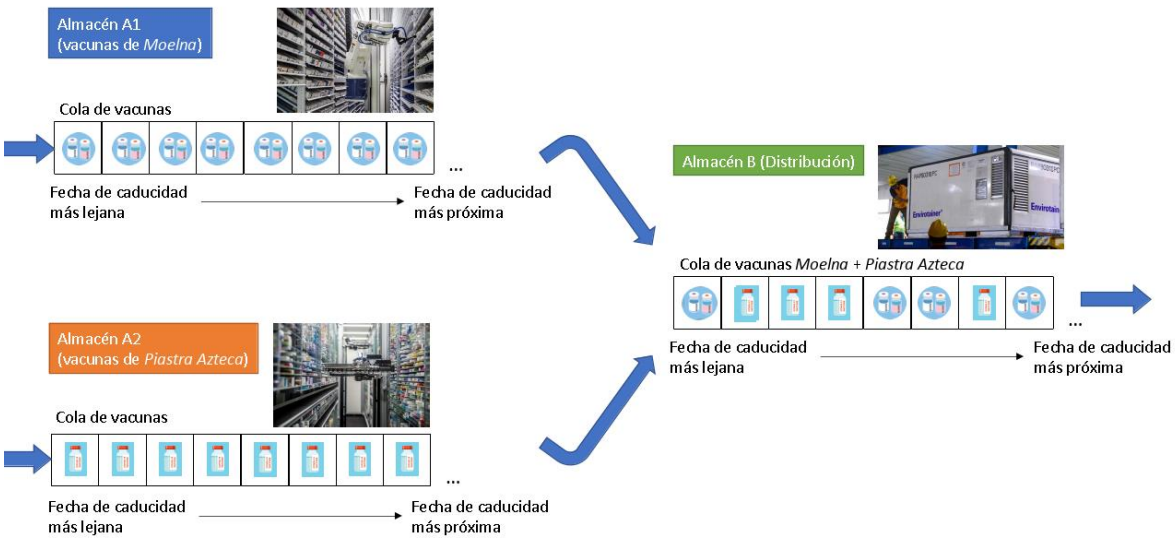
Material a entregar

- Proyecto compilable con el código fuente necesario para la práctica → **Repositorio SCV**
- Documentación del código → **Aula Virtual**

Introducción

Uno de los retos en la gestión de las vacunas contra la COVID-19 es evitar que los viales alcancen su fecha de caducidad antes de ser administrados. La Consellería de Sanitat necesita desarrollar un sistema que permita seleccionar qué viales deben ser distribuidos a los ambulatorios de tal manera que se seleccionen siempre los que tengan fecha de caducidad más cercana.

Desde Consellería se han puesto en funcionamiento tres almacenes: dos almacenes de larga duración (almacenes A1 y A2), en los que se guardan las vacunas cuando llegan desde la fábrica, y otro de distribución (almacén B), en el que se preparan y almacenan las vacunas que se van a enviar a los puntos de vacunación en las próximas horas.



Estos tres almacenes están gestionados por un centro de coordinación que mantiene una base de datos con los lotes que hay en cada almacén y con los lotes que ya se han administrado.
Cada día, un número de lotes de vacunas del almacén B se distribuye a los centros de atención primaria y a los vacunódromos. A continuación, el centro de coordinación ordena que otra cantidad de lotes de vacunas pase de los almacenes A1 y A2 al almacén B para su distribución. Esta cantidad puede ser diferente a los lotes enviados a los puntos de vacunación.
Como hay vacunas que emplean diferentes tecnologías, sus periodos de caducidad difieren. Por tanto, no podemos administrarlas simplemente por orden de llegada. Por este motivo, el almacén A1 guarda las vacunas de la marca *Moelna* y el almacén A2 las vacunas de la marca *Piastra Azteca*. Cuando se recibe un cargamento de vacunas, se almacenan las de cada marca en su almacén y se guarda la información de los lotes en sendas colas en la base de datos. Los lotes de cada marca sí llegan ordenados por fecha de caducidad.
Cuando el centro de coordinación decide cuántos lotes deben pasar al almacén de distribución, se van cogiendo lotes de los dos almacenes A1 y A2, de tal manera que todas las vacunas que quedan en estos depósitos caducan más tarde que las vacunas que pasan al almacén B. Todas estas vacunas pasan al almacén B y se guardan sus datos en la base de datos, en una cola.
Cuando se envían las vacunas a los puntos de vacunación la información se guarda en la base de datos en una cuarta cola, indicando además la fecha de administración.
El objetivo de esta práctica es desarrollar un programa de gestión que permita realizar las siguientes acciones:

- Dar de alta los lotes de vacunas nuevos en los almacenes A1 y A2 cuando lleguen desde las fábricas.
- Traspasar al almacén de distribución (almacén B) vacunas para su posterior envío a los centros de vacunación. Para reducir el riesgo de que las vacunas caduquen, se deben escoger siempre las vacunas con fecha de caducidad más cercana, con independencia de la marca.
- Retirar del almacén B las vacunas distribuidas a los centros de vacunación, registrando su fecha de administración.

En Aula Virtual encontraréis ficheros correspondientes a envíos de lotes de vacunas de las fábricas a los almacenes para que podáis llevar a cabo las pruebas correspondientes.

Ejercicios

Se debe escribir un programa, llamado “`Pr7.cpp`”, que se irá modificando de acuerdo a lo indicado en las siguientes tareas y ser creará un proyecto pr7.dev para compilar y comprobar los avances de cada tarea. Recuerda que es necesario llevar el seguimiento de revisiones del código generado utilizando el sistema de control de versiones que se viene utilizando durante todo el curso en la asignatura. En esta práctica se debe trabajar en la carpeta “Pr7” del repositorio que ya tienes creado para la asignatura. Al finalizar cada tarea, se deberán registrar los cambios realizados al igual que se ha realizado en las 6 prácticas anteriores. En esta práctica ya no se muestran en el guion avisos para indicar cuándo y cómo se debe realizar este proceso, pues se supone que el estudiante ya debe ser capaz de realizarlo de manera autónoma. No realizar el correcto registro de versiones será valorado negativamente en la calificación de la práctica.

De la misma forma, a medida que realicéis la implementación correspondiente a cada tarea debéis realizar pruebas de comprobación de vuestro trabajo, utilizando para ellos los ficheros de datos disponibles en Aula Virtual. El código de estas pruebas debe quedar reflejado en los diferentes registros en el sistema de control de versiones.

Tarea 1

En este proyecto es necesario realizar comprobaciones relativas a fechas. Copia los ficheros de la clase Fecha de la práctica 5 para incluirlos en este proyecto. Sobrecarga los operadores < y > para que devuelvan un booleano indicando si una fecha es anterior/posterior a otra. Comprueba el correcto funcionamiento antes de continuar usando el fichero Pr7.cpp.

Tarea 2

Implementa una clase Lote de acuerdo con la siguiente especificación.

TAD Lote

Dominio: Representa los lotes de vacunas gestionados, con los siguientes atributos:

- Identificador del lote, de tipo Cadena
- Fecha de caducidad
- Administrado
- Fecha de administración
- Marca, de tipo entero (0 → No definida; 1 → Moelna; 2 → Piastra Azteca)

Operaciones:

- *CrearLote() -> Lote*
- *AsignaValores(Lote,Cadena,Fecha,Booleano,Fecha,Entero) -> Lote*
- *AsignaFechaAdministracion(Lote,Fecha) -> Lote*
- *DevuelveFechaCaducidad(Lote) -> Fecha*
- *DevuelveFechaAdministración(Lote) -> Fecha*
- *DevuelveAdministrado(Lote) -> Booleano*
- *DevuelveIdentificador(Lote) -> Cadena*
- *DevuelveMarca(Lote) -> Entero*
- LeerLote(Lote,FlujoEntrada) -> Lote
- *GuardaLote(FlujoSalida,Lote) -> Booleano*
- *MostrarLote(Lote)*

Sea l ∈ Lote, a ∈ Entero, c ∈ Cadena, i ∈ Entero, fec1,fec2 ∈ Fecha, f ∈ FlujoSalida

CrearLote()	*Crea un lote. Debería iniciar todos los campos de información del evento correctamente: la cadena a "<Sin asignar>", las fechas a un valor correcto de fecha, por ejemplo 1/1/1970, y la marca a 0.*

AsignaValores(l,c,fec1,a,fec2,i)	*Asigna valores a la información del lote l: c=identificador, fec1=fecha de caducidad, a=administrado, fec2=fecha de administración, i=marca.*
AsignaFechaAdministración(l,fec1)	*Marca el lote como administrado y asigna valor a la fecha de administración del lote l.*
DevuelveFechaCaducidad(l)	*Devuelve la fecha de caducidad del lote l.*
DevuelveFechaAdministracion(l)	*Devuelve la fecha de administracion del lote l.*
DevuelveAdministrado(l)	*Obtiene si el lote l ha sido administrado.*
DevuelveIdentificador(l)	*Obtiene el identificador del lote l.*
DevuelveMarca(l)	*Obtiene la marca del lote l.*
LeerLote(f,l)	*Lee del flujo de entrada los datos de un lote y los guarda en el lote l. Devuelve un booleano indicando si se han podido leer los datos.*
GuardaLote(f,l)	*Guarda en el flujo de salida f los datos del lote l.*
MostrarLote(l)	*Muestra por pantalla la información del lote l.*

Requisitos para la implementación de la clase

- Cuando los datos de los lotes se guardan en un fichero, se empleará formato CSV usando como separador el punto y coma (;). Es decir, cada lote será guardado en una única fila con los datos separados por punto y coma (;) y en el orden en que aparecen definidos en el TAD. Esto debe tenerse en cuenta al leer y guardar los datos de un lote.
- El método *MostrarLote()* se debe implementar sobrecargando el operador "<<". Debe mostrarse el nombre del fabricante. Si el lote no ha sido administrado debe mostrarse "NO ADMINISTRADO" en lugar de la fecha. Ejemplos:

```
Lote 3f5gysl45
Fecha de caducidad: 2/7/2022
Fecha de administración: 27/4/2022

Lote az435fasg
Fecha de caducidad: 25/9/2022
Fecha de administración: NO ADMINISTRADO
```

Implementa el código necesario para comprobar el correcto funcionamiento de la clase.

Tarea 3

Implementa una clase Almacén de acuerdo con la siguiente especificación.

TAD Almacén

Dominio: Representa un almacén de vacunas, con los siguientes atributos:

- Nombre
- Vacunas, de tipo cola de lotes

Operaciones:

- *CrearAlmacén(Nombre) -> Almacén*
- *AlmacenarLote(Almacén,Lote) -> Almacén*
- *DevuelveFechaPrimerLote(Almacén) -> Fecha*
- *ExtraeLote(Almacén) -> Lote*
- *DevuelveStock(Almacén) -> Entero*
- *AlmacénVacío(Almacén) -> Booleano*
- *GuardaAlmacen(FlujoSalida, Almacén)*
- *MuestraEstadoAlmacen(Almacén)*

Sea a ∈ Almacén, n ∈ Cadena, l ∈ Cola, l ∈ Lote, i ∈ Entero, fec ∈ Fecha, f ∈ FlujoSalida

CrearAlmacén(n)	*Crea un almacén. Debería iniciar la cola del almacén vacía y fijar el nombre en el constructor.*
AlmacenarLote(a,l)	*Inserta un lote al final de la cola del almacén.*
DevuelveFechaPrimerLote(a)	*Devuelve la fecha de caducidad del primer lote del almacén.*
ExtraeLote(a)	*Saca de la cola el primer lote y lo devuelve.*
DevuelveStock(a)	*Devuelve el número de elementos en la cola.*
AlmacénVacío(a)	*Indica si el almacén está vacío o no.*
GuardaAlmacen(a,f)	*Guarda la información de los lotes almacenados en el flujo de salida.*
MuestraEstadoAlmacen()	*Muestra por pantalla los lotes con caducidad más cercana y más lejana del almacén y el número de lotes.*

Requisitos para la implementación de la clase

- Para el almacenamiento de la cola de lotes en la clase Almacén, emplead el tipo de datos std::queue de la biblioteca STL de C++.
- La clase std::queue de la STL de C++ no permite recorrer los elementos. Por ese motivo, para poder guardar los datos de un almacén es necesario ir vaciando la cola a medida que se toman los elementos. Tanto el método GuardaAlmacen() como el método MuestraResumenAlmacen() deben llevar el calificador const. Por ello no se puede modificar la cola de la clase.
- En el método GuardaAlmacen() cada lote deberá almacenarse siguiendo el mismo formato CSV definido en la clase Lote.
- El método MuestraEstadoAlmacen() debe implementarse sobrecargando el operador "<<". A continuación, se muestra el formato que debe mostrarse, suponiendo que el almacén se llame "Almacén de larga duración A1".

```
*****************************************
Datos del Almacén:
== Almacén de larga duración A1 ==
Cantidad de lotes almacenados: 32

** Lote más antiguo: **
```

```
Lote he3gysl45j
Fabricante: Moelna
Fecha de caducidad: 3/6/2022
Fecha de administración: NO ADMINISTRADO

** Lote más reciente: **
Lote az43fsg5d
Fabricante: Piastra Azteca
Fecha de caducidad: 15/9/2022
Fecha de administración: NO ADMINISTRADO
*****************************************
```

De nuevo, implementa el código necesario para comprobar el correcto funcionamiento de la clase.

Tarea 4

Para cargar las vacunas desde un fichero, disponéis del operador “>>” que vuelca los datos de un fichero a un objeto de la clase Almacen. En el programa Pr7_prueba_istream.cpp tenéis un ejemplo de uso. La salida esperada es el siguiente bloque de texto seguido de una copia del fichero Envio1.csv:

```
*****************************************
Datos del Almacén:
== Almacen A1 ==
Cantidad de lotes almacenados: 31

** Lote más antiguo: **
Lote: 7d2a08ddb2c6
Fabricante: Piastra Azteca
Fecha de caducidad: 14/06/2022
Fecha de administración: NO ADMINISTRADO

** Lote más reciente: **
Lote: a622fd69cef1
Fabricante: Moelna
Fecha de caducidad: 15/06/2022
Fecha de administración: NO ADMINISTRADO
*****************************************
```

Si habéis implementado correctamente las clases anteriores, este programa debería compilar y funcionar sin ningún cambio. Además, podeís extenderlo para añadir trantas pruebas como consideréis necesarias antes de continuar.

Tarea 5

Implementa una clase Distribuidos que sea una clase derivada/heredada de la clase Almacén.

TAD Distribuidos

Dominio: Representa el conjunto de lotes de vacunas ya distribuidos en los centros de vacunación. Tiene los mismos atributos que la clase Almacén. Además, tiene las siguientes operaciones

Operaciones:

- *CrearDistribuidos(Cadena) -> Distribuidos*

- *DistribuirLote(Distribuidos,Lote,Fecha) -> Distribuidos*
- *DevuelveFechaUltimoLote(Distribuidos) -> Fecha*
- *DevuelveCantidad Distribuidos (Distribuidos) -> Entero*

Sea v ∈ Distribuidos, n ∈ Cadena, l ∈ Lote, fec ∈ Fecha

CrearDistribuidos(n)	*Crea un conjunto de lotes distribuidos. Deberá inicializar los atributos de la clase base de la manera adecuada, usando como nombre la cadena recibida.*
DistribuirLote(v,l,f)	*Añade un lote al final de la cola de lotes distribuidos, fijando su fecha de distribución y poniendo a true el atributo correspondiente del lote.*
DevuelveFechaUltimoLote(v)	*Devuelve la fecha de administración del último lote distribuido.*
DevuelveCantidadDistribuidos (v)	*Devuelve el número de lotes de vacunas que han sido guardados como distribuidos.*

Comprueba el correcto funcionamiento de esta clase.

Tarea 6

Una vez añadidas las 3 Clases necesarias (Lote, Almacen y Administradas) y comprobada su correcta implementación, borraremos las instrucciones utilizadas para probar las clases del fichero pr7.cpp y realizaremos la implementación del sistema de gestión de almacenes de vacunas.

La aplicación debe mostrar la cantidad de vacunas disponibes en los tres almacenes, las vacunas administradas y las acciones disponibles por medio de un menú:

```
Almacenes de larga duración:
* Almacén A1 -> 17 lotes de vacunas.
* Almacén A2 -> 12 lotes de vacunas.
Almacén de distribución:
* Almacén B -> 3 lotes de vacunas.
Vacunas administradas:
* Lotes de vacunas admiminstrados -> 45.
* Último lote administrado el 4/5/2022.

Seleccione una acción:

    1. Recibir vacunas.
    2. Enviar vacunas a almacén B.
    3. Enviar vacunas a puntos de vacunación.
    4. Mostrar estado almacenes.
    5. Guardar estado almacenes.
    0. Salir.
```

El programa solicitará al usuario que introduzca una opción hasta que el usuario pulse 0. En cada opción se realizará lo siguiente:

1. Recibir vacunas. Se solicitará el nombre de un fichero con la información de los lotes recibidos. Los lotes se insertarán en el mismo orden en que se encuentran en el fichero, en los almacenes A1 o A2, en función del fabricante.

2. Enviar vacunas al almacén B. Se solicitará al usuario la cantidad de vacunas a enviar. Se irá extrayendo la vacuna con fecha de caducidad más baja disponible en los almacenes de larga duración (A1 y A2) y se insertará en el almacén de distibución (almacén B). Al terminar la acción se mostrará un mensaje con la información del envío:

```
================================
Envío con 12 lotes:
7 lotes de la marca Moelna
5 lotes de la marca Piastra Azteca
La fecha de caducidad del último lote extraido es el 24/7/2022
================================
```

Esta acción debe funcionar correctamente si uno de los almacenes o ambos se quedan sin vacunas. En el caso en que ambos almacenes se queden sin stock se deberá mostrar un mensaje indicando que el envío no se ha podido completar y mostrando la cantidad de vacunas realmente enviado:

```
================================
Envío con 9 lotes
7 lotes de la marca Moelna
2 lotes de la marca Piastra Azteca
La fecha de caducidad del último lote extraido es el 24/7/2022
** AVISO: Envío incompleto. **
** Se solicitaron 12 lotes. Stock disponible: 9. **
================================
```

3. Enviar vacunas a puntos de vacunación. Se pedirá una cantidad de vacunas, informando sobre la cantidad máxima disponible en el almacén B, y una fecha para el envío. El número de lotes solicitado (o el máximo disponible) se extraerá del almacén B y se registrarán como Vacunas Administradas indicando la fecha de administración.
4. Mostrar estado almacenes. Para cada almacén, mostrará los lotes con caducidad más cercana y más lejana del almacén y el número de lotes disponibles, de acuerdo con el formato especificado en la Tarea 3.
5. Guardar estado almacenes. Se solicitará un nombre de sesión, que será una cadena sin espacios, y se guardarán los lotes de los 3 almacenes y de las vacunas administradas en 4 ficheros CSV. Los nombres de los ficheros seran el nombre de sesión seguido de un sufijo indicando el almacen. Por ejemplo, si el nombre de sesion es “4MAYO22”, los ficheros se llamarán “4MAYO22_A1.csv”, “4MAYO22_A2.csv”, “4MAYO22_B.csv”, “4MAYO22_VA.csv”.
0. Salir. Preguntará si se quiere guardar la sesión. Si la respuesta es afirmativa se ejecutará el guardado como en la opción 5.

Detalle de implementación. Para gestionar los tres almacenes de vacunas se deben emplear objetos de la clase Almacen. Para gestionar el listado de vacunas ya distribuidas se usará un objeto de la clase Administradas.

Pruebas de funcionamiento. Debéis tomar nota de las pruebas realizadas para comprobar el funcionamiento de la versión final y añadirlas a la documentación con Doxygen. Estas pruebas deberán consistir en una secuencia de opciones de menú y de entradas de parámetros que permitan comprobar que el programa funciona correctamente. La documentación de las pruebas debe aparecer en la página principal de la documentación. Para ello, al final del fichero Pr7.cpp podéis añadir un bloque de comentario con la orden \mainpage.

```
/** \mainpage Sistema de distribución de vacunas
*
*  Este texto aparecerá en la página principal de la documentación.
*  Debéis colocar este bloque de comentario al final del fichero Pr7.cpp
*  y modificarlo para indicar qué acciones habéis realizado para comprobar
*  la versión final del programa, especificando una secuencia de entradas
*  de usuario/a.
*
*  Consultad la documentación de Doxygen para formatear adecuadamente el texto
*  ( https://www.doxygen.nl/manual/index.html ).
*
*/
```

Tarea 7: Generación de documentación

El código realizado deberá estar correctamente documentado con Doxygen, de acuerdo con la guía de estilo. Una vez realizada la versión final del programa, se deberá ejecutar Doxygen para generar toda la documentación definitiva. La documentación deberá ser generada exclusivamente en formato en *html* (utilizar como referencia, convenientemente personalizado para esta práctica, el archivo de configuración de Doxygen disponible en el Aula Virtual). Solo la documentación deberá subirse a Aula Virtual (como archivo comprimido) una vez concluido el trabajo.