Fundamentos de Programación
PRÁCTICA. Semestre 20221.
UOC Cinema

Formato y fecha de entrega

Se debe entregar la práctica (PR) antes del día **21 de diciembre a las 23:59 h**. Para ello, deberéis entregar un archivo en formato **ZIP** de nombre ***logincampus*_pr** en minúsculas (donde *logincampus* es el nombre de usuario con el que entráis en el campus). El ZIP debe contener:

- El **workspace CodeLite** entero, con todos los archivos que se piden.

Para reducir el tamaño de los archivos y evitar problemas de envío que se pueden dar al incluir ejecutables, es necesario eliminar lo que genera el compilador tal y como se explica en xWiki. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente (las subcarpetas *Menu* y *Test* son las que contienen todos los archivos binarios que genera el compilador).

Es necesario realizar la entrega en el Registro de Evaluación Continua (REC) del aula de teoría.

Presentación

En esta práctica se presenta un caso que hemos trabajado durante algunas de las PEC de este semestre, en concreto, en las PEC 4, 6 y 8. El contexto, pues, será el mismo, pero ampliándolo con la gestión de películas, cines y las compras que se realizan por parte de los espectadores.

Se quiere, también, que empecéis a poner en juego una competencia básica para un programador: la capacidad de entender un código ya dado y saber adaptarlo a las necesidades de nuestro problema. A tal fin, se le facilita gran parte del código, en el que existen acciones y funciones métodos muy similares a las que se piden. Se trata de aprender mediante ejemplos, una habilidad muy importante a desarrollar.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Analizar un enunciado y extraer los requisitos tanto de tipos de datos como funcionales (algoritmos).
- Poner en práctica, de forma integrada, los conceptos fundamentales de algorítmica y programación C estudiados en la asignatura:
    - Aplicar correctamente las estructuras alternativas cuando haga falta.
    - Aplicar correctamente las estructuras iterativas cuando sea necesario.
    - Utilizar correctamente los tipos de datos estructurados (*Tuplas*).
    - Utilizar correctamente el tipo de datos *Tabla*.
    - Aplicar correctamente el concepto de *Modularidad*.
    - Aplicar correctamente los algoritmos de búsqueda y recorrido.
- Analizar, entender y modificar adecuadamente código existente.
- Profundizar en el uso de un IDE, en ese caso CodeLite.
- Hacer pruebas de los algoritmos implementados en lenguaje C.

Recursos

Para realizar esta actividad tenéis a vuestra disposición los siguientes recursos:

- Materiales en formato web de la asignatura.
    - xWiki
    - FAQ de Fundamentos de Programación
    - Ejercicios resueltos de semestres anteriores
- Foros de la asignatura (laboratorio y teoría).

Criterios de valoración

Cada ejercicio tiene asociada la puntuación porcentual sobre el total de la actividad. Se valorará tanto la corrección de las soluciones como su completitud.

- **Los ejercicios en lenguaje C, deben compilar** para ser evaluados. En tal caso, se valorará:
    - Que funcionen correctamente.
    - Que pasen los juegos de prueba
    - Que se respeten los criterios de estilo (ver la *Guía de estilo de programación en C* que tenéis en la Wiki).
    - Que el código esté comentado. En caso necesario (cuando el código no sea trivial) se valorará la presencia de comentarios explicativos.
    - Que las estructuras utilizadas sean las más adecuadas en el lenguaje C.

Descripción del proyecto

Este semestre nos han pedido crear una aplicación para *UOCCinema,* una cadena de cines que necesita gestionar sus espacios, salas, sesiones y películas exhibidas. En las PEC 4, 6 y 8 habéis ido definiendo ciertas variables y programando pequeños algoritmos relacionados con esta aplicación. En el código que os proporcionamos como base para la realización de esta práctica podréis encontrar estas variables ya conocidas de las PEC y algunos de los algoritmos.

Las acciones que queramos tener programadas serán las siguientes:

- Leer, mediante un menú, los datos correspondientes a un *cine y una película.*
- Cargar los datos desde archivos y también almacenarlos.
- Realizar búsquedas, aplicar filtros y obtener datos estadísticos.
- Gestionar las compras de tickets para ver las películas.

Además, aunque esta primera versión será una aplicación online de pedidos, queremos que todas estas funcionalidades queden recogidas en una **API** (*Application Programming Interface*), lo que nos permitirá en un futuro poder utilizar esta aplicación en diferentes dispositivos (interfaces gráficas, teléfonos móviles, tabletas, web, ...).

Estructuración del código

Junto con el enunciado se os facilita un proyecto CodeLite que será el esqueleto de la solución de su práctica. En la práctica debéis trabajar con este código, al que no será necesario añadir ningún archivo más. Únicamente completaremos los tipos, funciones y/o acciones que se nos vayan indicando en el enunciado. A continuación se dan algunas indicaciones del código proporcionado:

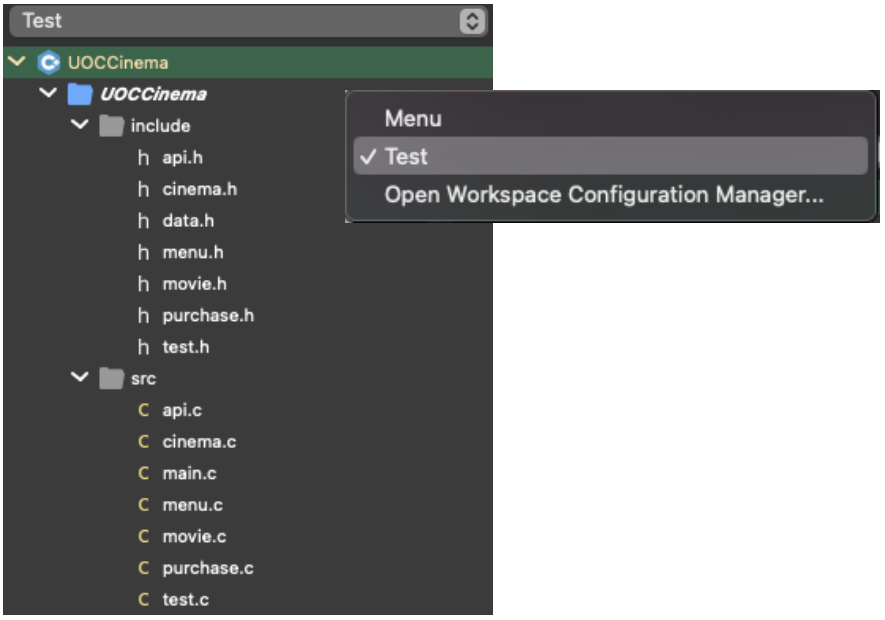
main.c

Contiene el inicio del programa. Está preparado para funcionar en dos modos diferentes, en modo *Menu* y en modo *Test*. Para gestionar qué modo debe ejecutar la aplicación se utilizan los parámetros pasados en el programa desde el entorno de ejecución. Estos parámetros se corresponden con los parámetros definidos en la función principal*:*

```
int main(int argc, char **argv)
```

Para que funcione en modo *Menu* no es necesario pasar ningún parámetro a la aplicación y, para que funcione en modo *Test*, es necesario pasar el parámetro "-t".

Este funcionamiento ya está implementado y el proyecto de *CodeLite* se ha configurado con dos modos de configuración *Menu* y *Test*. Podéis cambiar el modo de ejecución utilizando el desplegable *Configuration Manager* del editor *CodeLite:*



- La aplicación, en **modo *Menu***, nos muestra por el canal de salida estándar un menú que permite al usuario interactuar con la aplicación: realizar altas y bajas de cines, salas, sesiones, películas, etc. También obtener información, datos estadísticos y demás funcionalidades relacionadas con los datos de la aplicación.

- La aplicación en **modo *Test*** ejecuta un conjunto de pruebas sobre el código para asegurar que todo funciona correctamente. El resultado de estas pruebas se mostrará una vez finalizada la ejecución por el canal de salida estándar.

  Inicialmente, muchas de estas pruebas fallarán, pero, una vez realizados todos los ejercicios, todas deberían pasar correctamente. Es importante que os **aseguréis que vuestro programa, una vez completados todos los ejercicios, pasa la totalidad de las pruebas incluidas en el proyecto**.

data.h

Se definen los tipos de datos que se utilizan en la aplicación. Aunque se podrían haber separado en los diferentes archivos de cabecera, se han agrupado todos para facilitar la lectura del código.

cinema.h / cinema.c

Contienen todo el código que gestiona cines, salas y sesiones.

movie.h / movie.c

Contienen el código que gestiona las películas.

purchase.h / purchase.c

Contienen el código que gestiona las compras de tickets que los espectadores hacen para ver una película en una sesión de una sala de un cine.

menu.h / menu.c

Contienen todo el código para gestionar el menú de opciones que aparece cuando se ejecuta en modo *Menu*.

api.h / api.c

Contienen los métodos (acciones y funciones) públicos de la aplicación, lo que sería la API de nuestra aplicación. Estos métodos son los que se llaman desde el menú de opciones y los que utilizaría cualquier otra aplicación que quisiera utilizar el código.

test.h / test.c

Contienen todas las pruebas que se pasan en el código cuando se ejecuta en modo Test.

Enunciado

Hemos hecho una toma de requisitos con la empresa *UOCCinema* y hemos identificado los campos necesarios en la **estructura de datos de las películas**. El conjunto de campos de la tupla *tMovie* queda resumida en la siguiente tabla:

Campo	Descripción	Tipo / validación
movieId	Identificador de la película.	De tipo ***tMovieId***. El identificador debe estar entre 1 y 50.
title	Título de la película	Cadena de, como máximo, 15 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un nombre compuesto, utilizaremos el guión bajo ('_') como separador de las palabras.
duration	Duración de la película	De tipo ***tTime*** (tupla que contiene las horas y minutos de duración)
rate	Calificación	Será un enumerado ***tMovieRate*** con los siguientes valores posibles: G_RATED, PG_RATED, PG13_RATED, R_RATED, NC17_RATED. El significado es el siguiente: • G_RATED: General Audiences (para todos los públicos) • PG_RATED: Parental Guidance Suggested (algunos contenidos podrían no ser apropiados para niños) • PG13_RATED: Parental Strongly Cautioned (algunos contenidos podrían no ser apropiados para menores de 13 años) • R_RATED: Restricted (los menores de 17 años requieren acompañamiento adulto) • NC17_RATED: Adults only (no recomendado por menores de 17 años)
income	Ingresos	Real que indica los ingresos (en euros) recaudados por la película. Debe ser 0 o positivo.

Respecto a la **estructura de datos de los cines**, esta es la relación de campos de la tupla *tCinema*:

Campo	Descripción	Tipo/validación
cinemaId	Identificador del cine	De tipo ***tCinemaId.***
type	Tipo de cine	Enumerado que puede ser PREMIERE, RELEASE, INDIE, OTHER. Indica si el cine es de estreno, de reestreno, cine independiente o de otro tipo.

name	Nombre del cine	Cadena de, como máximo, 15 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un nombre compuesto, utilizaremos el guión bajo ('_') como separador de las palabras.
city	Ciudad donde se encuentra el cine	Cadena de, como máximo, 15 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un nombre compuesto, utilizaremos el guión bajo ('_') como separador de las palabras.
openingTime	Horario de apertura del cine	De tipo tTime (tupla que contiene la hora y minuto de apertura)
closingTime	Horario de cierre del cine	De tipo tTime (tupla que contiene la hora y minuto de cierre)
screens	Salas de proyección	Es una tabla de pantallas (tScreen). En cines multisala, puede haber varias.

Los cines pueden tener una o más de una **sala de proyección**. Representaremos la sala de proyección con una variable *tScreen*, que consta de los siguientes campos:

Campo	Descripción	Tipo / validación
screenId	Identificador de la sala	De tipo ***tScreenId.***
capacity	Capacidad de la sala	Entero positivo que indica cuántos espectadores caben en la sala.
rows	Número de filas	Entero positivo que indica cuántas filas de butacas hay en la sala.
seatsPerRow	Número de butacas	Entero positivo que indica cuántas butacas hay en cada fila
sesiones	Sesiones de proyección	Es una tabla de sesiones asociadas a una sala.

Cada sala puede tener varias **sesiones**, en distintos horarios. La tupla *tSession* se encargará de representar una sesión, con estos campos:

Campo	Descripción	Tipo / validación
sessionId	Identificador de la sesión	De tipo ***tSessionId.***

movieId	Identificador de la película	De tipo ***tMovieId***
time	Hora de la sesión	De tipo tTime (tupla que contiene la hora y minuto de inicio de la sesión)
busySeats	Número de butacas ocupadas	Entero positivo que indica cuántas butacas están ocupadas en la sesión
seats	Patio de butacas de la sala para esta sesión	Tabla bidimensional de asientos donde se representa su estado (tSeatStatus)

Cada sesión lleva el registro de las butacas ocupadas. Tened presente que sólo tienen sentido las posiciones de la tabla *seats* que están en el rango *rows x seatsPerRow* (valores que vienen dados por la sala donde se programa la sesión).

Por último, la entidad que representa la **compra de butacas** de una sesión en una sala de cine para ver una película es la tupla *tPurchase*. Los campos son:

Campo	Descripción	Tipo / validación
cinemaId	Identificador del cine	De tipo ***tCinemaId.***
screenId	Identificador de la sala	De tipo ***tScreenId.***
sessionId	identificador de la sesión	De tipo ***tSessionId.***
purchasedSeats	Número de butacas adquiridas en esta compra	Entero positivo
assignedSeats	Indica si la compra comporta asignación de asientos.	Booleano. Si es cierto, el campo ***seats*** tiene contenido. Si es falso, estará vacío.
seats	Vector con la relación de asientos asignados.	Cada posición es de tipo ***tSeat*** (par fila, asiento). Este vector tendrá ***purchasedSeats*** posiciones llenas si ***assignedSeats*** es cierto. De lo contrario, estará vacío.
price	Real	Contiene el precio en euros de esta compra.

Aparte de los tipos estructurados, revisad la definición de constantes existentes en el código. Será necesario utilizar algunas de ellas para resolver los ejercicios.

La empresa *UOCCinema* ya nos ha dado el visto bueno al análisis entregado, y podemos empezar a realizar los cambios. Para implementar estos cambios, tal y como ya hemos comentado, partiremos del código fuente que nos han proporcionado, e iremos añadiendo el código necesario.

Ejercicio 1: Definición de tipos y operaciones básicas con tablas [15%]

Responded los siguientes apartados:

a) [5%] Se pide que completéis el tipo de datos ***tMovie***, provisionalmente definido en el archivo *data.h.* Completad la definición que hay en el código enunciado como convenga. Para ello, fijaros en la descripción y el nombre exacto del nombre de los campos que se han especificado anteriormente en el enunciado.

IMPORTANTE

- Dado que estamos definiendo la estructura de datos, que es la base de la práctica, debéis empezar obligatoriamente por este apartado. Es importante que respetéis el orden indicado de los campos y su nomenclatura.

  Una vez hecho esto, descomentad la definición de la variable TYPEDEF_COMPLETED que encontraréis al principio del archivo *data.h* antes de continuar con el resto de ejercicios.

b) [5%] Se pide que completéis, en el archivo *movie.c*, la acción ***movieTableAdd*** que, dada una tabla tipo *tMovieTable* y una película tipo *tMovie*, añada la película dentro de la tabla. Si no hay espacio en la tabla para añadir la película, deberéis devolver el código ERR_MEMORY en la variable retVal. De lo contrario, devolved OK.

c) [5%] Se pide que completéis, en el archivo *movie.c*, la acción ***movieTableDel*** que, dada una tabla de tipo *tMovieTable* y una película de tipo *tMovie*, borre la película de la tabla. Tened presente que deberéis evitar dejar agujeros en la tabla, desplazando, si es necesario, los elementos de la tabla para ocupar el espacio que ocupaba la película eliminada.

Ejercicio 2: Entrada interactiva de datos [10%]

Completad la acción ***readMovie*** (*menu.c*) para que lea por teclado una película. En la entrada de datos debéis validar que los valores que vayan tomando los campos estén dentro del rango permitido por cada uno de ellos, según se ha descrito anteriormente.

Leed la duración de la película directamente en el formato HH:MM. Se deben leer todos los datos a excepción del campo *income*, que debéis inicializar a cero.

Esta acción se puede probar en modo Menu, utilizando la opción Manage *Movies* > *Add Movie*.

NOTA: Fijaros en la lectura del tipo tCinema, que hace una lectura de campos muy similar a la que se pide en este ejercicio.

Ejercicio 3: Programación de una película en una sala de cine [15%]

Para programar una película en una sala de cine, deben darse diversas condiciones previas. Por ejemplo, la película debe haberse dado de alta previamente y el cine debe tener alguna sala donde añadir la sesión para proyectar la película. Esto es lo que hace, precisamente, la acción que pedimos que completéis a continuación:

a) [10%] Completad el código que falta en la acción **addSession** (*api.c*). En este punto del código disponéis de estas variables:
   - prevSession, la última sesión que está programada en la sala.
   - prevMovie, la película que está programada en la prevSession.

   Vuestro código debe añadir una sesión a continuación de prevSession, teniendo en cuenta que la nueva sesión debe comenzar 20 minutos después de que finalice la película anterior. Usad *timeCpy* y *timeAdd* para operar con los tiempos (podéis ver sus cabeceras en *data.h*)

   Esta acción se puede probar en modo Menu, utilizando la opción *Manage Cinemas > Add Session*.

Una vez programada una película en una sala, debe tenerse en cuenta de cara a ciertas operaciones. Por ejemplo, si queremos borrar una película de la tabla de películas, deberemos comprobar que no esté programada. Si lo está, no vamos a permitir borrarla. Esto es lo que se hace en *movieMenu* (*menu.c*) cuando se quiere borrar una película. La comprobación se hace llamando a una función, que se desea que completéis:

b) [5%] Completad el código de la función **isMovieProgrammed** (*api.c*) que, a partir de un objeto tAppData y una película tMovie, comprueba si la película está programada en alguna sesión de alguna sala de cine. La función deberá devolver cierto si la encuentra y falso en caso contrario.

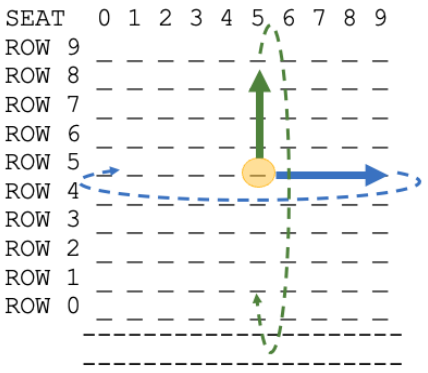
   Esta acción se puede probar en modo Menu, utilizando la opción *Manage Movies > Delete Movie*.

Ejercicio 4: Compra de entradas por una sesión [15%]

Durante el proceso de compra de entradas se llevan a cabo diferentes acciones. En primer lugar, se debe seleccionar un cine, una sala y una sesión, en la que se proyecta una determinada película. A continuación, es cuando se debe buscar una ubicación en el patio de butacas donde haya localidades libres y, por último, efectuar la compra marcando estos asientos como ocupados. Este proceso es el que se hace en la acción *processPurchase* (api.c), cuya lógica, se apoya en dos acciones que se pide que trabajéis a continuación.

a) [10%] Completad la acción **findSeats** (*cinema.c*) que, a partir de una sesión, dimensiones del patio de butacas (filas y asientos por fila) y el número de asientos deseados (entero positivo), devuelva la fila y asiento seleccionados. El resultado debe entenderse como el primero de los asientos reservados en caso de que el número de asientos deseados sea más de uno.

   La acción debe empezar a buscar asientos desde el medio del patio de butacas, buscando N localidades consecutivas libres. La búsqueda debe realizarse desde el medio hacia la derecha (incrementando el número de asiento) y, si no se encuentran localidades libres en esta mitad, se empieza a buscar desde el primer asiento de la fila hasta la mitad de la misma. En cuanto a las filas se procede igual, es decir, partiendo de la fila de en medio y avanzando hacia el final de la sala (incrementando el número de fila). Si no se encuentran localidades en esta mitad del patio de butacas, se comienza desde la primera fila avanzando hacia la fila central. Esquemáticamente:



b) [5%] Completad la acción **assignSeats** (*cinema.c*) que, dada una sesión, una fila y asiento iniciales y un número de asientos, hace la reserva de todas las localidades indicadas. La reserva se hace marcando cada uno de los asientos como PURCHASED en el mapa de asientos *seats* de la *tSession*. Actualizad el campo *busySeats* para reflejar la ocupación de la sesión.

NOTA: Para probar este ejercicio podéis realizar compras de entradas con la opción de Menu *Manage purchases > Purchase tickets for movie*. Posteriormente, podéis comprobar las localidades asignadas en el patio de butacas con la opción de Menú *Manage purchases > View session occupation*.

Ejercicio 5: Escritura de datos por pantalla [5%]

Otra forma de verificar la asignación de butacas en una compra es obteniendo la relación de los tickets para acceder a la sala. A tal efecto, se pide que implementéis la acción **printTickets** (*menu.c*) que, a partir de una compra, del cine, la sala, la sesión y la película, muestre por pantalla todos los tickets de cada una de las localidades adquiridas en la compra. Por cada ticket se debe mostrar: nombre del cine, título de la película, número de sala, hora de inicio de la sesión, fila y asiento.

NOTA: Para probar este ejercicio podéis utilizar la opción de Menu *Manage purchases > Print purchase tickets*.

Ejercicio 6: Funciones de copia y comparación [10%]

Un problema que nos encontramos con los tipos estructurados es que muchos de los operadores que tenemos definidos con los tipos básicos de datos, como los de comparación (==, !=, <, >, ...) o el de asignación (=), no funcionan para los nuevos tipos que nos creamos.

Por eso, a menudo se hace necesario definir acciones o funciones que nos den estas funcionalidades. Por ejemplo, ya hemos visto que para asignar una cadena de caracteres no lo hacemos con el operador de asignación normal (=), sino que debemos recurrir a la función **strcpy**. Lo mismo ocurre en las comparaciones, donde en vez de utilizar los operadores normales (==, !=, <, >, ...) utilizamos **strcmp**.

Se pide:

a) [5%] Completad, en el archivo *movie.c*, la acción **movieCpy** que permite copiar todos los datos de una estructura *tMovie* a otra.

b) [5%] Completad, en el archivo *movie.c*, la función **movieCmp** que, dadas dos películas m1 y m2, nos devuelve:

-1 si m1<m2 0 si m1=m2 1 si m1>m2

El orden de las películas vendrá dado por el valor de sus campos en el orden de prioridad siguiente:

1. Título de la película (ascendente)
2. Duración de la película (ascendente)
3. Calificación (ascendente)
4. Ingresos (ascendente)

Esto significa que si el título de m1 es “Cyborg” y el de m2 “Doraemon” , consideraremos que m1<m2 En caso de que los títulos sean iguales, será necesario comprobar la duración para desempatar Si las duraciones también son iguales, habrá que comprobar la calificación y, en última instancia, los ingresos. Si todos los datos son iguales, querrá decir que m1=m2

Nota: Para realizar este ejercicio podéis utilizar el método *strcmpUpper*, que compara strings sin tener en cuenta mayúsculas/minúsculas, así como *timeCmp* para comparar los tiempos.

Ejercicio 7: Filtros sobre colecciones de elementos [15%]

Entendemos por filtro una acción que, sobre un conjunto de elementos, selecciona algunos que cumplan una determinada condición.

En este caso, se desea que implementéis tres filtros:

a) [5%] Implementad, en *cinema.c* la acción **cinemaTableFilterByType** que, a partir de una tabla de cines y un tipo de cine, devuelva (en una segunda tabla) los cines que cumplan que son del tipo indicado.

b) [5%] Implementad, en *cinema.c* la acción **cinemaTableFilterByMultiScreen** que, a partir de una tabla de cines devuelva (en una segunda tabla) los cines que sean multisala.

c) [5%] Implementad la acción **cinemaTableGetMultiscreenPremiereCinemas** en *cinema.c* que, a partir de una tabla de cines devuelva (en una segunda tabla) los cines que sean de estreno y que sean multisala simultáneamente. Para implementar este apartado es obligatorio utilizar las acciones de los apartados anteriores.

Ejercicio 8: Estadísticas [15%]

Como parte importante de la gestión de la compañía y, como herramientas de apoyo a la toma de decisiones, se quieren implementar una serie de acciones y funciones con fines estadísticos.

En concreto, se pide:

a) [5%] Implementad la función **cinemaTableComputeAverageOccupation** en *cinema.c*. Esta acción debe calcular la ocupación media de todos los cines. Como veréis, cada cine calcula su ocupación como la media de las ocupaciones de las diferentes sesiones de sus salas, en un cálculo en cadena del que sólo se os pide que hagáis el correspondiente al nivel más alto. Para implementar esta acción usad la acción *cinemaAverageOccupation* (*cinema.c*).

b) [5%] Implementad la acción **movieTableGetAvgDuration** *(movie.c)* que, dada una tabla de películas, devuelva una tupla tTime con la duración promedio de todas las películas de la tabla.

c) [5%] Implementad la acción **movieTableGetIncomePerRate** *(movie.c)* que, dada una tabla de películas, obtenga los ingresos acumulados organizados por calificación de la película (G_RATED, PG_RATED, PG13_RATED, R_RATED, NC17_RATE). Los ingresos deben devolverse en los correspondientes parámetros de salida de la acción.