PRÁCTICA 3

Formato y fecha de entrega

La entrega se debe efectuar en el apartado “Entrega y registro de EC” del aula de teoría antes del día **20 de diciembre de 2019 a las 23:59**.

Se debe entregar un fichero en formato ZIP que contenga un directorio **‘UOCFlix2019’** con el código resultante de los ejercicios.

Las entregas deben cumplir los siguientes puntos para poder ser evaluadas correctamente:

- Se debe entregar un único fichero en formato ZIP que contenga el directorio principal ‘UOCFlix2019’ (y que no contenga otros ficheros .zip dentro).
- El sistema y nombre de los ficheros del proyecto entregado debe seguir la estructura del código adjunto al enunciado de la práctica (no modificar ni el nombre de los ficheros, ni el de las carpetas y tampoco modificar la estructura de ficheros dentro del directorio principal).
- El código entregado **no debe presentar errores de compilación** para poder ser evaluado. Si una parte del código funciona correctamente pero algún ejercicio posterior provoca un error de compilación, se aconseja borrar o comentar dicha parte para que el resto del código compile y se pueda evaluar la práctica.
- Si la entrega contiene código comentado, se aconseja indicar mediante comentarios los motivos de esta decisión.
- **El código debe poderse ejecutar**. Puede que algunos test fallen, pero, como mínimo, el programa debe ser capaz de mostrar el resultado de las pruebas.
- Se deben entregar todos los ficheros .c y .h utilizados en el proyecto y deben estar presentes en la entrega en las carpetas correctas (src, tests, etc.).
- Se deben entregar los ficheros .workspace y .project que definen el proyecto completo de CodeLite.
- Se recomienda añadir dentro de la carpeta principal del proyecto un fichero de nombre README.txt especificando el sistema operativo utilizado.
- Los ficheros de test se deben entregar sin modificaciones. Si para alguna prueba durante el desarrollo han sido modificados, en la entrega se debe volver a la versión original.

El incumplimiento de cualquiera de los puntos indicados puede suponer un suspenso en la práctica.

Presentación

Durante las diferentes prácticas desarrollaremos un único proyecto, sobre el que iréis diseñando e implementando funcionalidades.

Trabajaremos siguiendo una metodología basada en pruebas. Esto significa que con el enunciado os facilitaremos un conjunto de pruebas funcionales de la aplicación. El objetivo es implementar el código necesario para pasar todas estas pruebas. Los ejercicios que se proponen son una guía para facilitar la resolución de las diferentes funcionalidades probadas. Es importante destacar el hecho que un código que pase todos los test no significa que sea un código 100% correcto. Existen otros criterios de evaluación que se detallan más adelante (apartado Criterios de Valoración). Además, los test no tienen porqué probar que se han realizado todos los puntos solicitados en el enunciado, es decir, pueden no cubrir el 100% del enunciado.

Durante la programación es normal que surjan dudas relacionadas con el lenguaje, tanto con la sintaxis como en la forma de trabajar con las estructuras de datos. Dirigir vuestras dudas sobre la programación al foro del laboratorio de la asignatura.

Objetivos

Los objetivos de esta práctica son:

- Trabajar con Tipos Abstractos de Datos (TAD).
- Implementar algoritmos búsqueda y ordenación.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.

Recursos

Para realizar esta práctica disponéis de los siguientes recursos:

Recursos Básicos

Laboratorio de Prácticas de Programación: Disponéis de un laboratorio asignado a la asignatura. En este laboratorio podéis resolver todas las dudas sobre la programación en lenguaje C, y la instalación y utilización del entorno de programación.

Material de Laboratorio: En el laboratorio, aparte del colaborador docente, disponéis de diferentes documentos que os ayudarán. En concreto, hay un manual de C y una guía de programación en C.

Transparencias de Síntesis: En los materiales de la asignatura (xWiki) tenéis las transparencias de síntesis. Para esta práctica os será de especial interés los apartados de «TAD en memoria dinámica», «Métodos de búsqueda» y «Métodos de ordenación».

Recursos Complementarios

Buscador web: La forma más rápida de obtener información actualizada sobre los parámetros, funcionamiento y ejemplos de utilización de cualquier función estándar de C (y en general de cualquier lenguaje), es mediante un buscador web.

Criterios de valoración

Para la valoración de los ejercicios se tendrá en cuenta:

- El código obtiene el resultado esperado dadas unas condiciones y datos de entrada diseñados para probar algunas situaciones de funcionamiento normal y otros casos especiales.
- El código entregado sigue la guía de estilo y las buenas prácticas de programación.
- Se separa correctamente la declaración e implementación de las acciones y funciones, utilizando los ficheros correctos.
- El código está correctamente comentado, valorando especialmente la utilización de comentarios en inglés.
- El grado de optimización en tiempo y recursos utilizados en la solución entregada.
- El código realizado es modular y estructurado, teniendo en cuenta la reutilización del código.
- Se realiza una gestión de memoria adecuada, liberando la memoria cuando sea necesario

Enunciado

En el contexto de una plataforma de series de televisión a la carta/online, se nos propone el desarrollo de un sistema de registro de las visualizaciones de los usuarios de ésta. El objetivo del sistema a desarrollar es facilitar el análisis estadístico de su uso, para poder ofrecer recomendaciones individualizadas a los usuarios de la plataforma y mejorar su oferta.

Las indicaciones que nos han dado son las siguientes:

- Se debe poder gestionar un conjunto de usuarios, almacenando su nombre.
- Se debe poder gestionar un conjunto de largometrajes, o episodios de las series de televisión ofrecidas en la plataforma.
- Se quiere registrar cada visualización individual por parte de cada usuario sobre cada episodio que visualice. Para cada visualización, se almacenará información relativa a ésta.
- Se desea conocer los episodios favoritos de los usuarios y extraer datos útiles usando esta información
- Se desea organizar la información y las estadísticas de la plataforma de manera que se puedan realizar búsquedas y obtener listas ordenadas de datos

En esta tercera práctica se propone implementar una búsqueda y ordenación de episodios (films) acorde con su popularidad, obtenida en base a sus visualizaciones y categorización como favorito en cada usuario.

Ejercicio 1: Obtener popularidad de los films [35%]

Junto con el enunciado se adjunta el código fuente de la práctica anterior donde se han añadido algunas funciones y también se han añadido los ficheros popularity.h y popularity.c. Estos ficheros contienen los tipos de datos y lo métodos para la gestión de las estadísticas, la búsqueda y la ordenación de los episodios según las visualizaciones de distintos usuarios y sus valoraciones.

Se pide:

1. Dada las definiciones de los ficheros popularity.h y favorite.h se propone:

    a. Implementa el método **popularity_getCntView()** que dado un episodio (tFilm) y un histórico de visualizaciones nos devuelva el **número de visualizaciones** que se han realizado para este episodio.

    b. Implementa el método **favorite_filmExists()** que dados un episodio y una pila de favoritos nos diga si este episodio está en la pila.

    c. Implementa el método **popularity_getCntUsrFavd()** que dado un film y una tabla de usuarios nos diga el número de usuarios que tienen este episodio entre sus favoritos.

2. Dada la definición de **tFilmStats** del fichero popularity.h implementa los métodos siguientes:

    a. Implementa el método **popularity_getFilmStats()** que dados un histórico de visualizaciones, una tabla de usuarios y un film, retorne las estadísticas de la serie en un tipo tFilmStats que contenga:

        - El número de visualizaciones de este episodio en el histórico
        - El número de usuarios que tienen este episodio entre sus favoritos.

    b. Implementa el método **popularity_init**(..) que dados un film y una estadística para este, retorne un elemento de tipo tPopularity rellenando sus campos. No debe referenciarse al film existente, debemos hacer una copia del film.

    c. Implementa el método **popularity_free**(..) que libere la memoria ocupada por un elemento de tipo tPopularity.

    d. Implementa el método **popularity_compare()** que dadas las estadísticas de dos films devuelva 1 si el primer episodio obtiene mejores estadísticas (gana). Un -1 si el segundo film obtiene mejores estadísticas y un 0 si los dos tienen el mismo

resultado en las estadísticas. Para determinar el orden se hará según los siguientes criterios:

1. Gana el episodio que aparece entre los favoritos de más usuarios.
2. En caso de empate gana el episodio que tenga más visualizaciones.
3. Si sigue el empate se considera que los dos episodios tienen la misma estadística/popularidad.

Una vez implementados los métodos especificados en este ejercicio deberían pasar correctamente los test referentes al ejercicio 1 [PR3_EX1_XX]. Los principales conceptos que han de quedar claros después de este ejercicio son:

- Métodos para recorrer una pila.
- Cálculos matemáticos básicos utilizando un TAD

Ejercicio 2: Gestión de las estadísticas de films [35%]

En el ejercicio anterior hemos obtenido una serie de resultados y estadísticas donde cada serie tiene sus propios datos. Las estadísticas se guardarán en una **lista doblemente enlazada** de popularidad **tPopularityList** donde los nodos serán de tipo **tPopularityListNode** y contendrán la serie y sus estadísticas **tPopularity** (el hecho de utilizar una lista forma parte de los requerimientos de la realización de este ejercicio académico).

Se pide que dada la definición de **tPopularityList** en el fichero popularity.h, implementar los métodos siguientes en el fichero popularity.c:

1. **popularityList_create()**: inicializa una estructura tPopularityList como una lista vacía.

2. **popularityList_insert()**: Añade un elemento a una lista en una posición concreta dada. En el caso que la posición sea incorrecta la función retornará un valor ERR_INVALID_INDEX, si hay algún error reservando espacio en memoria para el nuevo elemento devolverá un valor ERR_MEMORY_ERROR. Si no ha habido errores devolverá OK.

3. **popularityList_delete()**: elimina un elemento de la lista en una posición concreta dada. En caso que la posición sea incorrecta devolverá un valor ERR_INVALID_INDEX, si la lista está vacía retornará un valor ERR_EMPTY_LIST. Si no hay errores retornará OK.

4. **popularityList_get()**: devuelve un elemento de la lista dada una posición concreta. En caso que la posición sea incorrecta retornará un valor NULL.

5. **popularityList_empty()**: indica si la lista que se le pasa está vacía (true) o, por el contrario, contiene elementos (false).

6. **popularityList_free()**: elimina todos los elementos de la lista.

Una vez implementados todos los métodos especificados en el ejercicio deberían pasar correctamente todos los test referentes al ejercicio 2 [PR3_EX2_XX]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Definición de una **lista doblemente encadenada**

Ejercicio 3: Búsqueda y ordenación de films por popularidad [30%]

En el trabajo con listas es habitual realizar recorridos en ella, búsquedas, o realizar cierta ordenación. En este apartado se pide que implementéis los siguientes métodos:

1. **popularityList_getPosByFilm()**: dada una lista de popularidad de films, que puede estar desordenada, devuelve la posición del elemento tPopularity en la lista (1 si es primero, 2 si es segundo, etc.). Si no encuentra el film en la lista o está vacía, devolverá -1.

2. **popularityList_bubbleSort**(): dada una lista desordenada de elementos tPopularity la ordena según sus estadísticas, de manera descendente. Se propone la implementación del algoritmo **Bubble Sort** (**ordenación por intercambio**). La lista resultante tendrá el elemento tPopularity con mejor estadística en primera posición, el segundo en segunda posición y así sucesivamente.

   Para ello se propone en primer lugar implementar la función **popularityList_swap**(..) para intercambiar dos elementos de la lista según su posición. En caso de llamar la función con índices fuera de rango (valor superior al tamaño de la lista) la función devolverá ERR_INVALID_INDEX. En caso de realizar correctamente el intercambio devolverá OK.

Nota: se propone utilizar la función **popularity_compare(..)** como criterio de ordenación / comparación de mayor/menor entre elementos tPopularity, en base a su tipo tFilmStats.

Una vez implementados todos los métodos especificados en el ejercicio deberían pasar correctamente todos los test referentes al ejercicio 3 [PR3_EX3_XX]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Búsquedas dentro de una lista doblemente encadenada.
- Ordenación de una lista doblemente encadenada.