Práctica 1

Formato y fecha de entrega

Hay que entregar la Práctica antes del día **25 de abril a las 23:59**. Para la entrega hace falta que entreguéis un fichero en formato **ZIP** de nombre ***logincampus*_pr1** en minúsculas (donde *logincampus* es el nombre de usuario con el que hacéis login en el Campus). El ZIP tiene que contener:

- Workspace CodeLite entero, con todos los ficheros que se piden.

Para reducir la medida de los ficheros, hay que eliminar lo que genera el compilador. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente.

Hay que hacer la entrega en el apartado de entregas de EC del aula de teoría.

Presentación

En esta práctica se presenta el caso en que trabajaremos durante este semestre. Habrá que utilizar lo que habéis trabajado en las primeras PECs, también la composición iterativa y la utilización de strings. También tendréis que empezar a poner en juego una competencia básica para un programador, la capacidad de entender un código ya dado y saberlo adaptar a las necesidades de nuestro problema. Con cuyo objeto, se os facilita gran parte del código, en el cual hay métodos muy similares a los que se os piden. Se trata de aprender mediante ejemplos, una habilidad muy necesaria cuando se está programando.

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 la programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Analizar un enunciado y extraer los requerimientos tanto de tipos de datos como funcionales (algoritmos)
- Aplicar correctamente la composición alternativa cuando haga falta
- Aplicar correctamente la composición iterativa cuando haga falta
- Utilizar correctamente tipo de datos estructurados
- Utilizar correctamente el tipo de datos tabla
- Aplicar correctamente el concepto de Modularidad
- Aplicar correctamente los esquemas de búsqueda y recorrido
- Analizar, entender y modificar adecuadamente código existente

Recursos

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

Básicos

- Materiales en formato web de la asignatura
- Laboratorio de C

Complementarios

- Internet: Una forma efectiva de encontrar información sobre cualquier duda sobre el lenguaje C es buscarlo a través de un buscador.

Criterios de valoración

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

- Los ejercicios en lenguaje C, tienen que compilar para ser evaluados. En tal caso, se valorará:

  - Que funcionen correctamente
  - Que se respeten los criterios de estilo (Ved la Guía de estilo de programación en C)
  - Que el código está comentado (preferiblemente en inglés)
  - Que las estructuras utilizadas sean las adecuadas

Descripción del proyecto

Durante este semestre realizaremos una aplicación para gestionar la biblioteca de la universidad. Las funcionalidades que queremos tener son:

- Gestionar **secciones**
  - Se identificarán por un código numérico (menor o igual que 10)
  - Para cada sección queremos almacenar un **nombre descriptivo** (máximo 100 caracteres)
  - Cada sección tendrá **subsecciones** identificadas también por un código numérico (menor o igual que 10).
- Gestionar **libros**
  - Tendrán como identificador su **ISBN**
  - Corresponderá a una **sección** y **subsección**
  - Almacenaremos
    - autor
    - título del libro
    - año de publicación
  - Queremos saber si está **disponible**
- Cargar los datos desde ficheros y también almacenarlos.
- Buscar los datos de un libro a partir del ISBN.
- Filtrar los libros de una sección.
- Obtener datos estadísticos de los libros.

Además, a pesar de que en esta primera versión será una aplicación en línea 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, ...).

Junto con el enunciado se os facilita un proyecto Codelite que será el esqueleto de la solución. A continuación se dan algunas indicaciones del código que os hemos dado:

main.c: Contiene el inicio del programa. Está preparado para funcionar en dos modos diferentes:

- **Menú**: Muestra un menú que permite al usuario gestionar los datos.
- **Test**: En este modo, se ejecutan un conjunto de pruebas sobre el código para asegurar que funciona. Inicialmente muchas de estas pruebas fallarán pero, una vez realizados todos los ejercicios, todas las de la PR1 tendrían que pasar correctamente.

Si se ejecuta normalmente, la aplicación muestra el menú. Para que funcione en modo test, hay que pasar el parámetro "-t" a la aplicación. Tal como se ha configurado el proyecto de Codelite, si ejecutáis en modo Debug se ejecutarán los tests, y si lo hacéis en modo Release, veréis el menú.

data.h: Se definen los tipos de datos que se utilizan a la aplicación. A pesar de que se podrían haber separado en los diferentes ficheros de cabecera, se han agrupado todos para facilitar la lectura del código.

books.h/books.c: contienen todo el código que gestiona la tabla para guardar los libros.

sections.h/sections.c: contienen el código que gestiona la tabla de secciones.

menu.h/menu.c: contienen todo el código para gestionar el menú de opciones.

apio.h/apio.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 nuestro código.

test.h/test.c: contienen todas las pruebas que se pasan al código cuando se ejecuta en modo Debug.

Todos los ejercicios de esta práctica consistirán en implementar acciones y funciones para manipular libros y secciones. La mayoría de los ejercicios

referencian a acciones y funciones ya existentes en el código proporcionado muy similares a las que se os piden, analizadlas y utilizadlas como base.

[15%] Ejercicio 1: Menú

Cuando trabajamos con aplicaciones que utilizan la línea de pedidos para comunicarse con el usuario, a menudo se hace necesario utilizar menús de opciones. El programa muestra la lista de opciones identificada por un número al usuario, de forma que el usuario pueda elegir la opción que le interesa introduciendo por teclado este identificador.

Tal como os hemos entregado el código, el menú que se presenta cuando se ejecuta la configuración "Release" no funciona correctamente. La lista de opciones aparece una única vez y sólo responde a las dos primeras opciones.

En este ejercicio se pide que analicéis el funcionamiento de las acciones y funciones del archivo menu.c y que completéis la acción **mainMenu** para que funcione:

- Es necesario que en función de la opción elegida (el entero introducido) se ejecute la acción que corresponda
- Y que una vez finalizada, se vuelva a mostrar el menú principal y a pedir al usuario que elija opción
- Excepto cuando la opción elegida es la de salir (en este caso el programa finalizará).

Debéis usar las constantes definidas en menu.h y las acciones ya existentes de menu.c que gestionan los submenús (bookMenu, secMenu o statsMenu). Podéis fijaros en la implementación de estas acciones para conseguir el funcionamiento del menú principal.

[15%] Ejercicio 2: Definición de la tupla libro, y lectura/escritura de sus campos

Los tipos estructurados de datos son una buena herramienta para representar objetos complejos como por ejemplo libros.

a) Completad en el fichero **data.h** la definición de la tupla **tBook** de forma que pueda almacenar:
   - un identificador (**ISBN**) que sea una cadena de **13** caracteres
   - el **año de publicación** (un entero)
   - un indicador de si está **disponible** (valor booleano)
   - una tupla **clasificación** que contenga dos campos: el **identificador de la sección principal** a la que pertenece (un carácter) y el **identificador de la subsección** (un carácter)
   - el **código del autor** (cadena de **3** caracteres sin espacios)
   - y el **título del libro** (cadena de como mucho **100** caracteres sin espacios)

     Usad **constantes** para escribir las longitudes fijadas de las cadenas.

b) Completad la función **readBook** en el fichero **menu.c** con la lectura de todos los campos de la tupla **tBook** (hace falta que trabajéis en modo Release y escojáis las opciones de menú “3 – Manage books” y “2 – Add book” para hacer pruebas).
   - Usad para cada campo el formato correspondiente de la función **scanf,** y para el campo **disponibilidad** usad un campo entero auxiliar y lo transformáis en booleano.

c) Completad en el fichero **books.c** la acción:

   void getBookStr(tBook book, int maxSize, char *str)

   que os permite pasar una estructura de tipo **tBook** a una cadena de caracteres.
   - Tomad como modelo la función **getSectionStr** del fichero **sections.c.** Los datos tienen que ir separados por espacios y en el mismo orden en que aparecen en el fichero **books.txt** que tenéis en la carpeta del workspace (ISBN año disponibilidad sección subsección autor título).

- Podréis comprobar el funcionamiento con la opción "**Add book":** tendrá que aparecer por pantalla una línea con los datos del libro acabado de introducir.

d) Completad en el fichero **books.c** la función:

tError getBookObject(const char *str, tBook *book)

Que os permite pasar una cadena de caracteres a una estructura de tipo tBook. Tomad como modelo la función **getSectionObject** del fichero **sections.c**.

- Los datos vendrán separados por espacios y en el mismo orden en que aparecen en el fichero **books.txt** que tenéis en la carpeta del workspace (*ISBN año disponibilidad sección subsección autor título*).
- Recordad que el título tiene que ser una cadena de caracteres sin espacios, podéis usar el guión bajo _ para separar palabras.

[20%] Ejercicio 3: Acciones que trabajan con la tupla

Un problema que nos encontramos es que muchos de los operadores que tenemos definidos para los tipos básicos de datos, como por ejemplo el de comparación o asignación, no funcionan para los nuevos tipos que nos creamos.

Por este motivo a menudo se hace necesario definir métodos que nos den estas funcionalidades. Por ejemplo, para asignar una cadena de caracteres no lo hacemos con el operador de asignación normal (=), sino que tenemos que recurrir a la función **strcpy**. Lo mismo pasa en las comparaciones, donde en vez de utilizar los operadores normales (==, !=, <, >, ...) utilizamos **strcmp**.

Fijaos como ejemplo en la función **section_cmp** y la acción **section_cpy** del fichero **sections.c** e implementar dentro del fichero **books.c:**

```
typedef struct {
        char isbn[MAX_ISBN_CODE];
        unsigned short int publicationYear;
        tBoolean isAvailable;
        tClassification classification;
        char authorCode[MAX_AUTHOR_CODE];
        char title[MAX_BOOK_TITLE];
} tBook;
```

a) La función **book_cmp** para comparar dos libros de forma que compara primero por **sección**, después por **subsección**, después por **autor**, después por **título** y, si coincide, por **ISBN**:

   int book_cmp(tBook b1, tBook b2)

   que devuelva 1 si b1 > b2, 0 si b1 = b2 y -1 si b1 < b2

b) La acción **book_cpy** para poder asignar a un libro la información de otro:

   void book_cpy(tBook *dst, tBook src)

   Hará una copia campo a campo de la información del libro src al libro pasado como parámetro de salida.

Podéis ejecutar en modo Debug para comprobar si pasáis los tests de la

comparación y la copia de libros.

[20%] Ejercicio 4: Manipulación de la tabla de libros

Las tablas son un tipo de datos que nos permite guardar un número variable de valores o tuplas. En el fichero **data.h** encontraréis definido el tipo **tBookTable** que define una tabla de libros.

Os pedimos:

a) Implementad la función:

tError bookTable_add(tBookTable *tabBook, tBook book)

que dado un parámetro de **entrada/salida** de tipo **tBookTable**, y un parámetro de **entrada** de tipo **tBook**, añada este libro al final de la tabla.

Esta función devolverá un valor **OK** en caso de que se haya podido añadir el nuevo libro a la tabla, o un valor **ERR_MEMORY** en caso de que no haya bastante espacio a la tabla para añadir el nuevo libro.

Para realizar este ejercicio tomad como ejemplo la implementación de la función **secTable_add** que podéis encontrar en el archivo s**ections.c.**

b) Implementad la función:

int bookTable_find(tBookTable tabBook, char *ISBN)

que dado un parámetro de entrada de tipo **tBookTable**, y un parámetro de entrada de tipo string, busque el libro que tiene este identificador ISBN en la tabla y nos devuelva el índice que le corresponde, o -**1 s**i no lo encuentra.

Para realizar este ejercicio tomad como ejemplo la implementación de la función **secTable_find** que podéis encontrar en el archivo **sections.c.**

c) Implementad la acción:

void bookTable_del(tBookTable *tabBook, tBook book)

que dado un parámetro de entrada/salida de tipo tBookTable, y un parámetro de entrada de tipo tBook, **elimina** de la tabla el libro que tiene el mismo ISBN que éste y desplaza todos los que están a continuación una posición atrás. En caso de que no haya ningún libro que coincida con el dado, no hará nada.

Para realizar este ejercicio usad las funciones de búsqueda y copia que ya habéis implementado en apartados anteriores. Tomad como ejemplo la función **secTable_del** que podéis encontrar en el archivo s**ections.c.**

[10%] Ejercicio 5: Persistencia de datos

La persistencia de los datos se garantiza gracias a los mecanismos que hacen que los datos no se pierdan cuando finaliza la ejecución del programa. Usaremos el paso a cadena de caracteres para poder escribir en un fichero nuestras tablas, y la lectura del fichero de texto y posterior lectura de las cadenas de caracteres para poder llenar nuestras tablas.

a) Implementad la función :

terror bookTable_save(tBookTable tabBook, const char *filename)

que dada una tabla de libros y un nombre de fichero, guarde todos los libros de la tabla en formato textual. Hay que utilizar en su código la acción **getBookStr.** La función que implementéis devolverá un valor **OK** si ha podido guardar los datos o un valor **ERR_CANNOT_WRITE** en caso de que, por algún motivo, no se pueda abrir el fichero de salida. Podéis utilizar como ejemplo la implementación de la acción **secTable_save** del fichero sections.c.

b) Implementad la función:

terror bookTable_load(tBookTable tabBook, const char *filename)

que dada una tabla de libros y un nombre de fichero, lea todos los datos de libros en formato texto del fichero y los añada a la tabla.

Usad la función **getBookObject.** La función que implementéis devolverá un valor **OK** si ha podido guardar los datos o un valor **ERR_CANNOT_READ** en caso de que, por algún motivo, no se pueda abrir el fichero de entrada. Podéis utilizar como ejemplo la implementación de la acción **secTable_load** del fichero sections.c.

[10%] Ejercicio 6: Filtro de libros

Una vez que tenemos todos los libros guardados en una tabla, en este ejercicio nos interesará encontrar los libros que cumplan unas ciertas condiciones.

Dado que no conocemos el número de libros que cumplirán las condiciones, utilizaremos una tabla de libros **tBookTable** para devolver los resultados de las búsquedas.

Se pide implementar la acción:

> ***void bookTable_filterBySection(tBookTable tabBook, int sectionID, tBookTable *result)***
>
> que dada una tabla de libros y un código de sección, nos devuelva todos los libros de esta sección.

Nota: Es interesante que aprovechéis las acciones anteriormente creadas para no repetir fragmentos de código.

[10%] Ejercicio 7: Cálculos estadísticos

Ya somos capaces de gestionar todos los libros de la aplicación. Los podemos añadir, eliminar o listar. Incluso podemos obtener un subconjunto siguiendo unos determinados criterios. Veremos ahora cómo obtener información global.

En este ejercicio se pide que implementéis las funciones:

a) **unsigned int bookTable_getOnLoanNumber(tBookTable tabBook)**

que dada una tabla de libros, nos devuelva la cantidad total de libros que no están disponibles porque están en préstamo.

b) **unsigned int bookTable_getAuthorNumber(tBookTable tabBook, char *author)**

que dada una tabla de libros, nos devuelva la cantidad total de libros del autor pasado por parámetro.