Fundamentos de Programación
PRÁCTICA 1. Semestre 20211.
Gestión de espacios de co-trabajo

Formato y fecha de entrega

Es necesario entregar la Práctica antes del día **10 de diciembre a las 23:59h**. Para la entrega se deberá entregar un archivo 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 debe contener:

- 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 la xWiki. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente (las subcarpetas *Menu* y *Test* son las que contienen todos los 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 el caso que hemos trabajado durante este semestre. Habrá que utilizar lo que se ha trabajado en las PEC's 1 a 8. 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 esta finalidad, se le facilita gran parte del código, en el que existen métodos muy similares a los que se 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 programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Practicar los conceptos estudiados en la asignatura.
- Profundizar en el uso de un IDE, en ese caso CodeLite.
- Analizar un enunciado y extraer los requisitos tanto de tipos de datos como funcionales (algoritmos).
- Aplicar correctamente las estructuras alternativas cuando sea necesario.
- 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.
- Realizar pruebas de los algoritmos implementados.

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 años 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 tiene en la Wiki).
  - Que el código esté comentado (preferiblemente en inglés). En caso necesario (cuando el código no sea trivial) se valorará la presencia de comentarios explicativos.
  - Que las estructuras utilizadas sean las adecuadas.

Descripción del proyecto

Este semestre nos han pedido crear una aplicación para *UOCoworking,* una compañía de gestión de espacios de co-trabajo. En las PECs se han 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 PECs y algunos de los algoritmos.

Recordad, por ejemplo, cómo definimos variables para almacenar datos de un centro de *coworking*.

Ahora en esta práctica os pediremos implementar la gestión de los usuarios de estos centros, los *coworkers*.

Y las acciones que queremos tener programadas serán las siguientes:

- Leer, mediante un menú, los datos correspondientes a un *coworker*.
- Cargar los datos desde archivos y también almacenarlos.
- Realizar búsquedas, aplicar filtros y obtener datos estadísticos.

Además, aunque esta primera versión será una aplicación online de pedidos, queremos que todas estas funcionalidades queden recogidas en una **API** (Application Programming Interface). Esto 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 facilita un proyecto CodeLite que será el esqueleto de la solución. En la práctica trabajaremos 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 que os hemos dado:

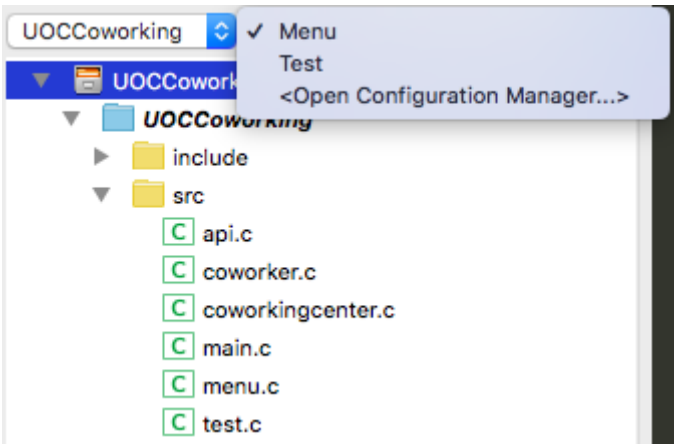
main.c

Contiene el inicio del programa. Está preparado para funcionar en dos modos diferentes, en modo *Menu* y en modo *Test*. Para gestionar en qué modo se debe ejecutar la aplicación, se utilizan los argumentos pasados en el programa desde el entorno de ejecución. Estos argumentos 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, el proyecto de *CodeLite se* ha configurado con dos modos de configuración *Menu* y *Test*. Se puede 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 gestionar los datos de nuestra 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 **asegurarse de que el 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.

coworker.h / coworker.c

Contienen todo el código que gestiona los co-trabajadores.

coworkingcenter.h / coworkingcenter.c

Contienen el código que gestiona los centros de co-trabajo.

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

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

Fijaros que la mayoría de los ejercicios solicitados en esta práctica referencian acciones y funciones ya existentes en el código proporcionado que son muy similares a las que se le piden. Analizadlas y utilitzadlas como base.

Enunciado

Tal y como hemos comentado en la presentación de la práctica, una de las estructuras de información que utilizaremos es la del co-trabajador.

Hemos realizado una toma de requisitos con la empresa *UOCoworking* y hemos identificado los campos que son necesarios en la estructura de datos del co-trabajador. Por cada campo hemos definido su nombre, su descripción, su tipo y, en algunos casos concretos, la validación a realizar a los valores introducidos por los usuarios. Os presentamos toda la información recogida en la siguiente tabla de **Campos del objeto Cotrabajador**:

Campo	Descripción	Tipo/validación
id	Identificador del cotrabajador	Hay que definirlo de tipo ***tCoworkerId***. Enter comprendido entre los valores 1 y 100 (ambos inclusive).
name	Nombre del cotrabajador	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.
surname	Apellido del cotrabajador	Cadena de, como máximo, 15 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un apellido compuesto, utilizaremos el guión bajo ('_') como separador de las palabras.
docNumber	Documento de identidad	Cadena de, como máximo, 9 caracteres alfanuméricos y sin espacios.
birthdate	Fecha de nacimiento	Tupla de tres enteros que representan el año, mes y día de nacimiento del cotrabajador. Estos valores se guardan en los campos de la tupla year, month y day respectivamente.
birthCity	Ciudad de nacimiento	Ciudad donde ha nacido el cotrabajador. 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.
civilState	Estado civil del cotrabajador	Será un valor enumerativo con las siguientes posibilidades: SINGLE, MARRIED, SEPARATED, DIVORCED, WIDOWED.
profile	Perfil profesional	Será un valor enumerativo con las siguientes posibilidades: FREELANCER, TELEWORKER, ENTREPRENEUR, OCCASIONAL, EMPLOYEE, PARTNER, OTHER.

bookingsCounter	Contador de reservas que ha hecho un cotrabajador	Es un valor entero, mayor o igual a 0
points	Contador de puntos que el cotrabajador ha ganado haciendo reservas.	Es un valor entero, mayor o igual a 0
satisfaction	Satisfacción del cotrabajador en el uso de los centros de cotrabajo	Es un real que indica el porcentaje, entre 0 y 100, de satisfacción

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

La empresa *UOCowork* 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 desarrollo de código necesario.

Ejercicio 1: Definición de datos [10%]

En este ejercicio se pide que complete el tipo de datos ***tCoworker,*** que está parcialmente definido en el archivo **data.h**, con los campos indicados en la tabla definida anteriormente (tabla *Campos del objeto cotrabajador*).

IMPORTANTE

- Dado que estamos definiendo la estructura de datos, que es la base de la práctica, debéis empezar obligatoriamente por este ejercicio. 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.

Ejercicio 2: Entrada interactiva de datos [10%]

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

Completad la implementación de la acción ***readCoworker***, que se encuentra en el archivo **menu.c**, que permite dar de alta de manera interactiva un nuevo cotrabajador. Esta acción es llamada desde la opción interactiva de añadir un nuevo cotrabajador “2) ADD COWORKER”. Para acceder es necesario elegir primero “3) MANAGE COWORKERS” del menú principal de la aplicación. Podéis utilizar estas opciones para comprobar que la entrada interactiva de datos funciona correctamente.

Los datos a pedir por cada trabajador son: id, name, surname, docNumber, birthDate (3 enteros), birthCity, civilState y profile. Los campos bookingsCounter, points y satisfaction no se pedirán por teclado sino que se inicializarán a cero.

Habrá que comprobar si los valores introducidos están dentro de los límites especificados en el ejercicio 1 y, si es así, será necesario que el parámetro de salida de la acción retVal contenga el valor OK y, en su defecto, el valor ERROR. Tomad como modelo la lectura de datos de centros de cotrabajo que se realiza en la acción **readCoworkingCenter** del mismo archivo menu.c.

Nótese que este ejercicio sólo se puede probar interactivamente dado que se trata precisamente de implementar la entrada/salida por teclado y pantalla. Por este motivo, no se proporcionan test asociados a este ejercicio.

```
======================
 Main Menu
======================
1) Load data from file
2) Save data to file
3) Manage coworkers
4) Manage coworking centers
5) View statistics
6) Exit
>> 3
======================
 Manage Coworkers
======================
1) List coworkers
2) Add coworker
3) Delete coworker
4) Clear all coworkers
5) Select urban coworkers in city
6) Get list of satisfied coworkers
7) Exit
>>
```

Ejercicio 3: Comparación de la estructura *coworker* [15%]

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 este motivo, a menudo se hace necesario definir métodos 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 solicita:

a) [10%] Completad, en el archivo **coworker.c**, la función ***strcmpUpper*** que compara dos strings (s1 y s2) pasados por parámetro pasándolos, previamente, a mayúsculas. La función no debe alterar los ajustes de entrada. Para ello, será necesario declarar variables locales de tipo string, copiar el contenido de los parámetros de entrada, transformar a mayúsculas y, finalmente, comparar los strings resultantes. La función deberá devolver:

-1 si s1<s2 0 si s1=s2 1 si s1>s2

Nota: Os podéis apoyar en el método **strcmp** de la librería **string.h** también en la función **convertToUpper** que encontrará en **coworker.c.**

b) [5%] Complete en el archivo **coworker.c** la función ***coworkerCmp*** que dados dos cotrabajadores c1 y c2, nos devuelve:

-1 si c1<c2 0 si c1=c2 1 si c1>c2

El orden de los cotrabajadores vendrá dado por el valor de sus campos con la prioridad:

1. Nombre (ascendente)
2. Apellido (ascendente)
3. Número de documento (ascendente)
4. Fecha de nacimiento (ascendente)
5. Ciudad de nacimiento (ascendente)
6. Estado civil (ascendente)
7. Perfil (ascendente)
8. Contador de reservas (ascendente)
9. Puntos acumulados (ascendente)
10. Satisfacción (ascendente)

Esto significa, si el nombre de d1 es Gerard y el de d2 Manel, consideraremos que d1<d2. En caso de que sean iguales, se comprobará el apellido para desempatar. Si también son iguales, habrá que comprobar el número de documento, y así sucesivamente. .. Si todos los datos son iguales, indicará que d1 = d2

Nota: Se pide que la comparación de cotrabajadores ignore las diferencias de mayúsculas y minúsculas que pueda haber entre cualquier campo de tipo string. Por tanto, apoyaros en la función **strcmpUpper** del apartado anterior cuando convenga. Disponéis también de la función de comparación de fechas **dateCmp** en coworker.c.

Ejercicio 4: Filtros basados en expresiones [15%]

Por medio de expresiones podemos construir condiciones que nos permitan seleccionar determinados elementos dentro de un conjunto mayor. En el caso de los cotrabajadores, este conjunto podría llegar a ser muy grande y con frecuencia será necesario recurrir a determinados filtros para focalizar la atención en un colectivo más pequeño.

Por ejemplo, desde el departamento de marketing, se nos puede pedir filtrar centros de cotrabajos y cotrabajadores para poder orientar, de forma más efectiva, diversas campañas y promociones, así como obtener listados de cotrabajadores que cumplen determinados niveles de satisfacción con fines comerciales.

Con este objetivo, se pide:

a) [10%] Completad la acción ***coworkerTableSelectCoworkers*** en el fichero **coworker.c** que, dada una mesa de cotrabajadores y una ciudad, seleccione a aquellos cotrabajadores susceptibles de escoger centros de cotrabajo de tipo urbano en esta ciudad. Desde el departamento de marketing de UOCowork se ha determinado que los cotrabajadores objetivo de este tipo de centros son, en su mayoría, los que tienen una edad comprendida entre 25 y 35 años (ambos incluidos), que están en estado SINGLE y que tienen un perfil de tipo OCCASIONAL, EMPLOYEE o PARTNER. Se excluirán los cotrabajadores que hayan nacido en la ciudad pasada por parámetro.

   Nota: Necesitaréis utilizar la función auxiliar **calculateAge** de coworker.c que, a partir de una fecha de nacimiento, calcula la edad respecto a una fecha "actual" (que estableceremos en el día 31 de diciembre de 2020).

b) [5%] Completad la acción ***coworkerTableSelectSatisfiedCoworkers*** en el archivo **coworker.c** que permite seleccionar a aquellos cotrabajadores con un índice de satisfacción superior al 85%. Sólo se tendrán en cuenta aquellos cotrabajadores que hayan realizado un número significativo de reservas (estableceremos este umbral en un mínimo de 10 reservas).

NOTA: Revisad las constantes de data.h y localizad aquellas que puedan serviros para resolver este ejercicio.

Ejercicio 5: Combinación de filtros [20%]

Realizar acciones y funciones que apliquen determinados filtros puede ser una herramienta útil en sí mismas, pero también como "piezas" de determinadas búsquedas y filtros más complejos. En este ejercicio le pediremos que combinéis filtros existentes de otros ejercicios y apartados, tanto de la práctica como de PEC's anteriores.

La empresa UOCowork tiene muchos usuarios itinerantes que viajan por diversas ciudades y que necesitan espacios de cotrabajo cuando están en ciudades que no son la suya propia.

Con este objetivo, se pide:

a) [10%] Implementad en el fichero **coworkingcenter.c** la acción ***coworkerRecommendation*** que, dado un cotrabajador y una tabla de centros de cotrabajo, devuelva la lista de centros recomendados en una ciudad dada y con una determinada puntuación mínima. Para ello, llamad a la función *coworkingCenterTableSelect* con un precio objetivo, distancia deseada y puntuación mínima (que, por tanto, también serán parámetros de coworkerRecommendation). Evitad recomendar centros en la misma ciudad de nacimiento del cotrabajador.

b) [10%] Implementad en **menu.c** la acción ***allCoworkersRecommendation***

que, a partir de una estructura tAppData, una ciudad, precio, distancia y puntos, muestre por pantalla, por cada cotrabajador, la lista de centros recomendados para él. Para ello, llamad iterativamente a la función coworkerRecommendation del apartado anterior con cada uno de los cotrabajadores devueltos por la función coworkerTableSelectCoworkers del ejercicio 4a.

Nota: Para mostrar un cotrabajador por pantalla podéis utilizar la acción *printCoworker*, y para mostrar un centro lo podéis hacer con *printCoworkingCenter*.

Ejercicio 6: Cálculos estadísticos [20%]

Tenemos todos los cotrabajadores guardados en una tabla de tipo tCoworkerTable y la aplicación ya nos permite gestionarlos (añadir, eliminar, etc). En este ejercicio nos interesará obtener algunos datos estadísticos relativos a ellos.

Se pide implementar en el archivo **coworker.c**, utilizando las acciones que necesite de este mismo archivo:

a) [10%] La función ***coworkerTableGetAvgPointsPerBooking***

   Que, dada una tabla de cotrabajadores calcule el número de puntos promedio que un cotrabajador consigue en cada reserva. Para ello, acumulad el número de reservas totales y el número de puntos totales antes de realizar el cálculo.

b) [10%] La acción ***coworkerTableGetMaxSatisfactionPerAgeInterval***

   Que, dada una tabla de cotrabajadores, calcule la satisfacción máxima para los siguientes intervalos de edad: menos de 30 años, entre 30 y 44, entre 45 y 59 y 60 años o más. Devolved la información mediante parámetros de salida. Si, para una determinada franja de edad, no hay datos que permitan realizar el cálculo, devolved un valor cero.

NOTA: Revise las constantes de data.h y localizad aquellas que puedan serviros para resolver este ejercicio.

Ejercicio 7: Operaciones con tablas [10%]

Se pide que implementéis la acción **coworkerTableDel** del archivo **coworker.c**, que dada una tabla de cotrabajadores y un cotrabajador, borra de la tabla el cotrabajador pasado por parámetro.

Tened presente que el borrado del cotrabajador no debe dejar ninguna posición inválida en la tabla. Por tanto, será necesario que ocupéis el espacio liberado por el cotrabajador desplazando una posición atrás todos los cotrabajadores que había después del cotrabajador borrado.

Nota: Para facilitar la solución, utilizad la función **coworkerTableFind** de coworker.c para localizar el índice del trabajador a borrar.