Prácticas de Programación
PR1 - 20211

Fecha límite de entrega: **24 / 10 / 2021**

Formato y fecha de entrega

La práctica debe entregarse antes del día **24 de octubre de 2021** a las 23:59.

Es necesario entregar un fichero en formato **ZIP**, que contenga una carpeta UOC20211 con el directorio principal de vuestro proyecto, siguiendo la estructura de carpetas y nombres de ficheros especificados en el enunciado de la práctica. No debe contener ningún fichero ZIP en su interior:

- ☐ Un fichero **README.txt** con el siguiente formato (ver ejemplo):

```
Formato:
Correo electrónico UOC
Apellidos, Nombre
Sistema operativo utilizado

Ejemplo:
estudiante1@uoc.edu
Apellido1 Apellido2, Nombre
Windows 10
```

- ☐ Los ficheros de prueba sin modificaciones.
- ☐ Los ficheros *.c y *.h resultantes de los ejercicios realizados.
- ☐ Los ficheros .workspace y .project que definen el espacio de trabajo y los proyectos de Codelite.
- ☐ Todos los ficheros deben estar dentro de las carpetas correctas (src, test, …).

La entrega debe realizarse en el apartado de entregas de EC del aula de teoría antes de la fecha límite de la entrega.

El incumplimiento del formato de entrega especificado anteriormente puede suponer un suspenso de la práctica.

Objetivos

- Saber interpretar y seguir el código de terceras personas.
- Saber compilar proyectos de código organizados en carpetas y librerías.
- Saber implementar un proyecto de código a partir de su especificación.

Criterios de corrección:

Cada ejercicio tiene asociada su puntuación sobre el total de la actividad. Se valorará tanto que las respuestas sean correctas como que también sean completas.

- No seguir el **formato de entrega**, tanto por lo que se refiere al **tipo y nombre de los ficheros** como al contenido solicitado, comportará una **penalización importante** o la cualificación con una **D de la actividad**.
- El código entregado **debe compilar para ser evaluado**. Si compila, se valorará:
    - Que **funcionen** tal como se describe en el enunciado.
    - Que obtenga el **resultado esperado** dadas unas condiciones y datos de entrada diseñadas (pruebas proporcionadas).
    - Que se respeten los **criterios de estilo y** que el código esté **debidamente comentado**. Se valorará especialmente el uso de comentarios en inglés.
    - Que les **estructuras** utilizadas sean las correctas.
    - Que se **separe correctamente la declaración e implementación** de las acciones y funciones, utilizando los ficheros correctos.
    - El **grado de optimización** en tiempo y recursos utilizados en la solución entregada.
    - Que se realice una **gestión de memoria** adecuada, liberando la memoria cuando sea necesario.

Aviso

Aprovechamos para recordar que **está totalmente prohibido copiar en las PECs** de la asignatura. Se entiende que puede haber un trabajo o comunicación entre los estudiantes durante la realización de la actividad, pero la entrega de esta debe que ser individual y diferenciada del resto. Las entregas serán analizadas con **herramientas de detección de plagio.**

Así pues, las entregas que contengan alguna parte idéntica respecto a entregas de otros estudiantes serán consideradas copias y todos los implicados (sin que sea relevante el vínculo existente entre ellos) suspenderán la actividad entregada.

Guía citación: https://biblioteca.uoc.edu/es/contenidos/Como-citar/index.html
Monográfico sobre plagio:
http://biblioteca.uoc.edu/es/biblioguias/biblioguia/Plagio-academico/

Observaciones

Esta PEC presenta el proyecto que se desarrollará durante las distintas actividades del semestre, que se ha simplificado y adaptado a las necesidades académicas.

En este documento se utilizan los siguientes símbolos para hacer referencia a los bloques de diseño y programación:

 Indica que el código mostrado es en **lenguaje** algorítmico.

 Indica que el código mostrado es en **lenguaje** C.

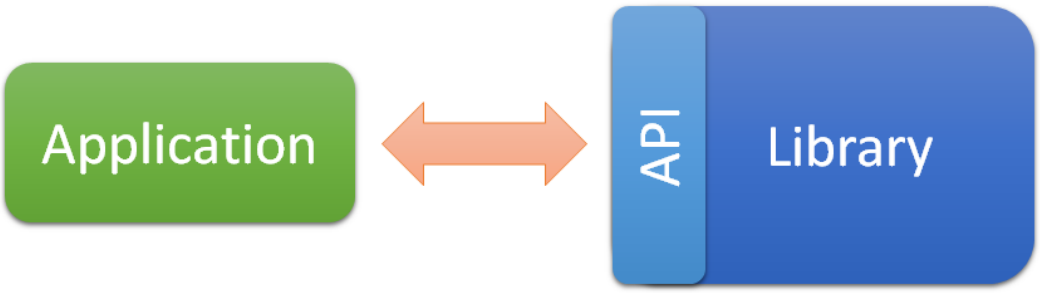
 Muestra la ejecución de un programa en **lenguaje** C.

Enunciado

En las PECs hemos trabajado de forma aislada partes del problema introducido en la PEC1. En las prácticas veremos como construir una aplicación más compleja que vaya incorporando de forma incremental lo que se va trabajando en las PECs.

Es habitual que las aplicaciones definan una API (Application Programming Interface) o interfaz de programación de aplicaciones. Básicamente se trata de una abstracción de nuestra aplicación, en la cual definimos los métodos y los datos, y permitimos que otras aplicaciones puedan interactuar con nuestra aplicación sin necesidad de saber como se han implementado estos métodos.

Además, encapsularemos todas las funcionalidades en una librería, que podrá ser utilizada por cualquier programa. En la siguiente figura se muestra la estructura de la práctica, en que tendremos una aplicación (ejecutable) que utilizará los métodos (acciones y funciones) de la API, implementada en una librería.



A nivel de código, lo que tendremos será un espacio de trabajo (Workspace) con dos proyectos:

- **Aplicación**: Será un proyecto igual al que utilizamos en las PECs, creado como un ejecutable simple.
- **Librería:** Será un proyecto de tipo “Static library”. En este caso, el resultado de la complicación será y entrelazado no produce un ejecutable, sino un fichero .a o .lib dependiendo del sistema operativo. Este fichero se puede utilizar desde otra librería o desde una aplicación. A diferencia de las aplicaciones, las librerías no implementan el método *main*.

Ejercicio 1: Preparación del entorno [20%]

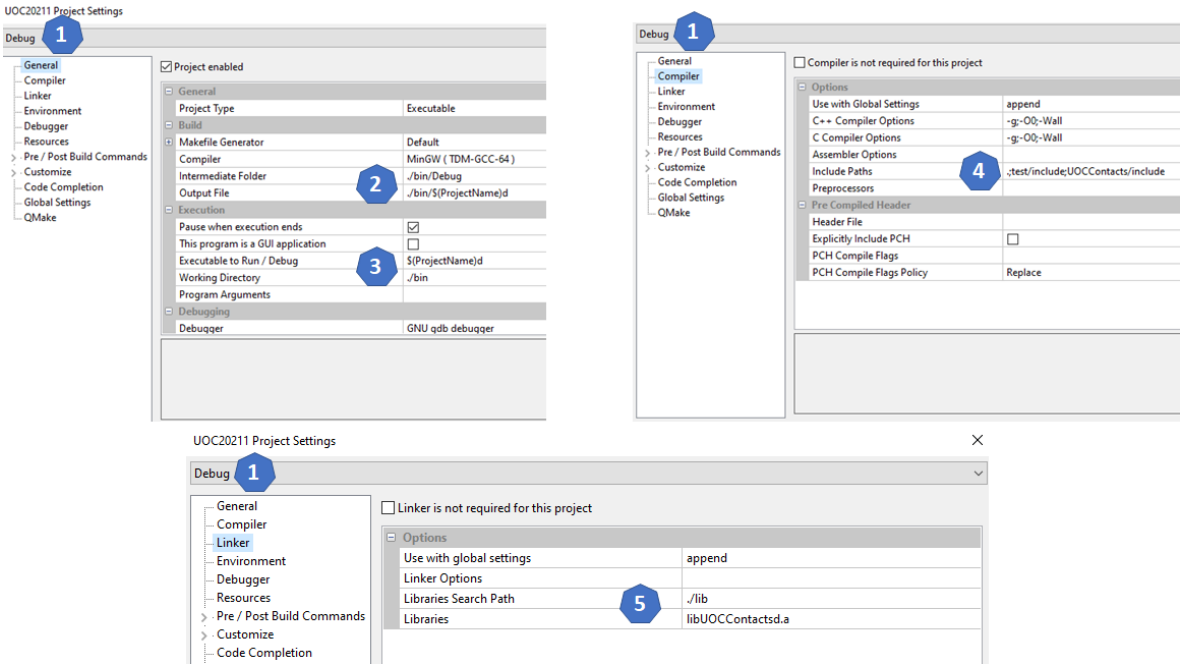
Junto con el enunciado se facilita un fichero de código con un espacio de trabajo (Workspace) que contiene dos proyectos. A continuación se detallan las principales características de cada uno de ellos:

- **UOCContacts**: Este proyecto corresponde a la librería donde iremos añadiendo toda la funcionalidad de la práctica.
    - El código se divide en delaraciónes (include) e implementaciones (src). Los ficheros **api.h** y **api.c** contienen la declaración de la API, y por lo tanto, los métodos que se accederán desde la aplicación.
    - Cuando se compila, debe ir a buscar los ficheros de cabecera en la carpeta include.
    - La librería se debe generar en la carpeta “lib” del Workspace.
    - El nombre de la librería añadirá una “d” cuando se compile en modo Debug.

- **UOC20211:** Este proyecto es nuestra aplicación. Se encarga de ejecutar las diferentes pruebas para verificar el correcto funcionamiento de la librería.
    - El código principal está en la carpeta src, y el código de las pruebas en la carpeta test, separando las declaraciones (include) y la implementación (src).
    - Cuando se compila, debe ir a buscar los ficheros de cabecera (*.h) tanto en la carpeta de las pruebas (test/include) como en la carpeta correspondiente de la librería (UOCContacts/include).
    - Cuando se hace el entrelazado (link), se debe indicar que vaya a buscar las librerías en el directorio lib del Workspace, y que incluya la librería generada por el proyecto anterior.

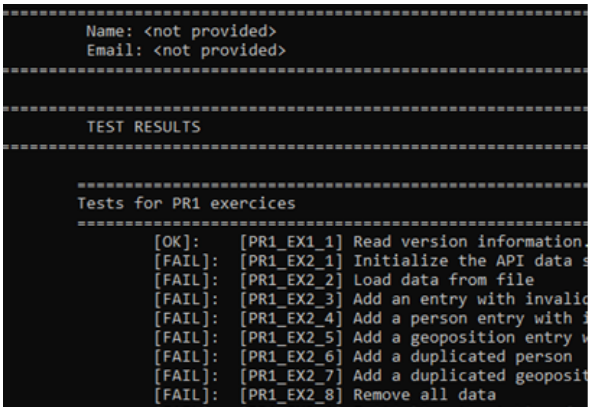
El objetivo de este ejercicio es tener el entorno proporcionado funcionando. Por tanto, será necesario que modifiquéis las opciones de los proyectos para que os funcione.

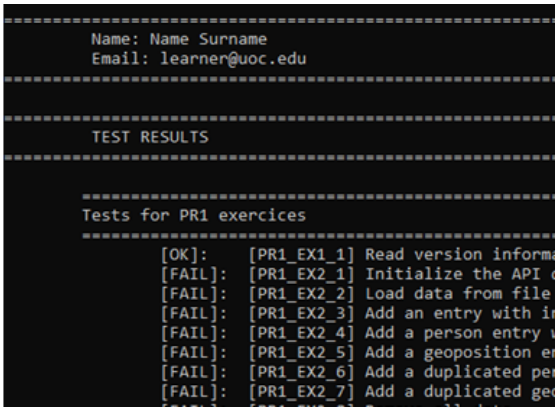
A continuación se muestra una guía de las opciones (settings) de los proyectos en donde se definen las características anteriores. Recordad que para acceder a las opciones del proyecto lo podéis hacer a través del menú contextual (botón derecho) y la opción Settings.



1. Permite cambiar entre la configuración de Debug y Release.
2. Define dónde se generan los ficheros resultantes
3. Define qué se ejecuta cuando se utiliza el botón de play de Codelite ( ). Se debe indicar el directorio de trabajo y el nombre de la aplicación, que será distinta en Debug y en Release.
4. Define en qué directorios se van a buscar los ficheros de cabecera.
5. Define en qué directorios se va a buscar las librerías y qué librerías se deben añadir para generar la aplicación.

Cuando se hayan seleccionado las opciones correctas, al ejecutar debéis ver el resultado de las pruebas. Inicialmente todas excepto la del ejercicio 1 aparecen como fallidas. También os saldrá que el nombre y el correo electrónico no se ha proporcionado (izquierda). Introducid los datos den el fichero README.txt del Workspace tal como se indica en el apartado de entrega, y se deberían incorporar.





Ejercicio 2: Entrada de datos [40%]

Hasta ahora hemos trabajado de forma aislada con los datos de las personas registradas en el sistema de salud (***tPopulation***) y los datos de geolocalización que se van capturando (***tGeolocationData***). Para facilitar la integración con la API, queremos agrupar todos los datos en una única estructura de datos (***tApiData***).

Estos datos se consultarán y manipularán a través de los métodos de la API (definidos en el fichero **api.h**), los cuales retornarán siempre un valor de tipo ***tApiError*** que indicará si se ha producido algún error o si la acción se ha ejecutado correctamente. Encontraréis los códigos de error definidos en el fichero **error.h** de la librería.

Utilizando como base las implementaciones en **memoria dinámica** de la PEC2:

a) Completa la definición del tipo de datos **tApiData** del fichero **api.h**, para que pueda guardar los datos referentes a personas y geolocalizaciones.

b) Implementa la función **api_initData** del fichero **api.c**, que inicializa una estructura de tipo ***tApiData*** dada. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.

c) Implementa la función **api_freeData** del fichero **api.c**, que elimina toda la información guardada en una estructura de tipo ***tApiData*** dada. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.

d) Implementa la función **api_addDataEntry** del fichero **api.c**, que dada una estructura de tipo ***tApiData*** y un nuevo dato en formato csv ***tCSVEntry***, guarda este nuevo dato dentro de la estructura ***tApiData***. Una entrada de datos puede ser de tipo “PERSON” o “GEOLOCATION” (podéis acceder al tipo mediante el método ***csv_getType***). Para cada dato, será necesario

comprobar que el formato sea correcto (asumimos que es correcto si el número de campos es el esperado), que tota persona referenciada en un dato de geolocalización haya sido registrada anteriormente y que no se añadan elementos duplicados. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.
E_INVALID_ENTRY_TYPE	El tipo de dato es incorrecto.
E_INVALID_ENTRY_FORMAT	El formato del dato no es correcto.
E_DUPLICATED_ENTRY	Se está intentando añadir un dato que ya existe.
E_PERSON_NOT_FOUND	No se ha encontrado ninguna persona con el documento de identidad proporcionado.

Nota: Podéis incorporar todos los ficheros de la PEC1 y PEC2 que creáis oportunos. Si queréis podéis utilizar las soluciones publicadas.

Ejercicio 3: Acceso a los datos [40%]

Con la finalidad de no exponer los tipos de datos internos de la API, se ha decidido que todos los intercambios de datos a través de la API se realizarán utilizando CSV. Recordad que cada entrada de un fichero CSV corresponde a un dato (fila, objeto, …), y que por lo tanto el fichero es un conjunto de datos. Utilizaremos el tipo ***tCSVData*** para intercambiar múltiples datos (por ejemplo listados), y el tipo ***tCSVEntry*** para objetos únicos. Teniendo en cuenta esto:

a) Implementa la función **api_findPerson,** del fichero **api.c**, que dada una estructura de tipo ***tApiData*** y un documento de identidad, nos retorne los datos de la persona en una estructura de tipo ***tCSVEntry***. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.
E_PERSON_NOT_FOUND	No se ha encontrado ninguna persona con el documento de identidad proporcionado.

b) Implementa la función **api_getPersonGeolocation,** del fichero **api.c**, que dada una estructura de tipo ***tApiData*** y un documento de identidad, nos retorne todos los datos de geolocalización disponibles para esta persona en una estructura de tipo ***tCSVData***, donde habrá una entrada (***tCSVEntry***) para cada dato de geolocalización. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.
E_PERSON_NOT_FOUND	No se ha encontrado ninguna persona con el documento de identidad proporcionado.

c) Implementa la función **api_removePerson,** del fichero **api.c**, que dada una estructura de tipo ***tApiData*** y un documento de identidad, elimina los datos de la persona y toda la información de geolocalización asociada. Los valores de retorno de esta función se detallan en la siguiente tabla:

E_SUCCESS	Operación ejecutada correctamente.
E_NOT_IMPLEMENTED	La funcionalidad aún no está implementada.
E_PERSON_NOT_FOUND	No se ha encontrado ninguna persona con el documento de identidad proporcionado.

Nota: Podéis asumir que un string en formato csv nunca superará los 2048 caracteres (2Kb) de longitud. Os pueden resultar de ayuda para realizar este ejercicio los siguientes métodos:

csv_init / csv_free	Inicializa / libera una estructura de tipo tCSVData
csv_parseEntry	Rellena una estructura de tipo tCSVEntry con la información contenida en un string en formato csv.
csv_addStrEntry	Añade a una estructura de tipo tCSVData una nueva entrada (tCSVEntry) a partir de un string en formato csv.
sprintf	Método similar a printf, pero que en lugar de mostrar una información formateada por pantalla, la guarda en un string.