Prácticas de Programación
PR3 - 20212

Fecha límite de entrega: **22 / 05 / 2022**

Formato y fecha de entrega

La práctica debe entregarse antes del día **21 de mayo de 2022** a las 23:59.

Es necesario entregar un fichero en formato **ZIP**, que contenga una carpeta **UOC20212** 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. Esta carpeta debe contener:

- 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. **Únicamente el último envío** dentro del periodo establecido será evaluado.

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 y prácticas** 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:

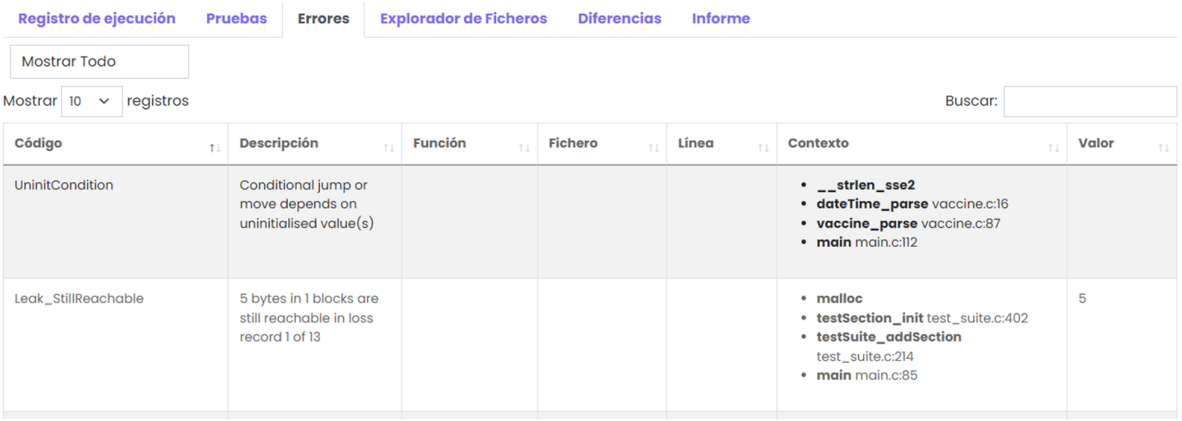
 Indica que el código mostrado es en **lenguaje** algorítmico.

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

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

Análisis dinámico

En esta actividad empezamos a utilizar memoria dinámica, que requiere que el programador reserve, inicialice y libere la memoria. Para ayudar a detectar memoria que no se ha liberado correctamente, o errores en las operaciones con punteros relacionadas, hay herramientas que ejecutan un análisis dinámico del programa. Una herramienta de código abierto muy empleada es Valgrind (https://valgrind.org/). La utilización de esta herramienta queda fuera del ámbito del curso, pero os facilitamos sus resultados como parte del análisis de la herramienta PeLP. Podéis acceder a los errores detectados por Valgrind en la pestaña de errores:



Para entender el significado de los **códigos de error**, podéis consultar el siguiente enlace, donde encontraréis ejemplos de código que os ayudarán a entender cuando se dan estos errores:

https://bytes.usc.edu/cs104/wiki/valgrind/

Enunciado

El punto de partida para esta práctica es el resultado de la PR1. El código facilitado con el enunciado incorpora todos los archivos necesarios de las actividades anteriores, con lo que veréis que todas las pruebas de la PR1 y PR2 pasan correctamente.

En esta práctica implementaremos la estructura de datos planteada en la PEC4 (con pequeñas modificaciones) y los métodos relacionados para poder gestionar las citas de vacunación:

```
type
      tAppointment = record
            timestamp : tDateTime;
            person : pointer to tPerson;
            vaccine : pointer to tVaccine;
      end record

      tAppointmentData = record
            elems : pointer to tAppointment;
            count : integer;
      end record

end type
```

Ejercicio 1: Gestión de las citas de vacunación [30%]

En el código proporcionado encontraréis los ficheros ***appointment.h*** y ***appointment.c*** con la definición de los tipos de datos anteriores y los principales métodos relacionados. En el fichero ***appointment.c*** implementad:

a) El método ***appointmentData_init*** que dada una estructura de tipo **t*Appointment*Data**, la inicialice correctamente. Inicialmente, no habrá ninguna cita programada.

b) El método ***appointmentData_insert*** que dada una estructura de tipo **t*Appointment*Data**, una fecha y hora **tDateTime**, el puntero a la definición de una vacuna **tVaccine** y el puntero a los datos de una persona **tPerson**, añada una nueva cita de vacunación para esta persona. Se debe insertar este registro de forma que:

i) Las citas de vacunación quedan ordenadas por día y hora.

ii) Para un mismo día y hora, las citas de vacunación quedan ordenadas por el documento de identidad de la persona asignada a la cita.

Nota: En los ficheros **date.h** y **date.c** encontraréis el método ***dateTime_cmp***, que os permite comparar dos fechas con hora, y dentro de la librería **string.h** tenéis el método estándar de C ***strcmp*** que permite comparar dos cadenas de caracteres.

c) El método ***appointmentData_remove*** que dada una estructura de tipo **t*Appointment*Data**, una fecha y hora **tDateTime**, y el puntero a la definición de una persona **tPerson**, elimine la cita de vacunación para esta persona en la fecha y hora proporcionadas. Si no existe, no hará nada.

d) El método **"recursivo"** ***appointmentData_find*** que dada una estructura de tipo **t*Appointment*Data**, el puntero a la definición de una persona **tPerson**, y una posición inicial, busque la primera cita de vacunación para esta persona, empezando por la posición indicada, y retorne su posición. Si se pasa una posición inicial 0, se buscará en tota la lista de citas. En caso que no exista ninguna cita de vacunación para esta persona a partir de la posición donada, retornará un valor -1. Si la posición inicial no existe, también retornará -1.

e) El método ***appointmentData_free*** que dada una estructura de tipo **t*Appointment*Data**, elimine toda la información que contiene.

Ejercicio 2: Integración en la API [40%]

Queremos incorporar la información sobre las citas de vacunación dentro de los datos de la api, y añadir los métodos necesarios para su gestión.

Se pide:

a) Modifica la definición del tipo **tHealthCenter** en el archivo **center.h** para que se incluyan las citas de vacunación **tAppointmentData**.

b) Modifica los métodos **center_init** y **center_free** del archivo **center.c**, para que se inicien y eliminen correctamente las citas de vacunación de los centros.

c) Implementa el método **api_addAppointment** del archivo **api.c**, que dada una estructura de tipo **tApiData**, el código postal de un centro de salud, el documento de identificación de una persona, el nombre de una vacuna, y una fecha y hora, añada una cita de vacunación con estos datos. Hay que tener en cuenta que si la vacuna requiere más de una dosis, habrá que añadir también las citas de vacunación para las siguientes dosis, en el mismo centro de salud. 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 existe ninguna persona con este documento de identidad.
E_VACCINE_NOT_FOUND	No existe ninguna vacuna con este nombre.
E_HEALTH_CENTER_NOT_FOUND	No existe ningún centro de vacunación con este código postal.

Nota: En los ficheros **date.h** y **date.c** tenéis el método **dateTime_addDay** que permite añadir un número de días a un objeto de tipo **tDateTime**.

d) Implementa el método **api_getPersonAppointments** del fichero **api.c**, que dada una estructura de tipo **tApiData** y el documento de identificación de una persona, nos retorne les cites de vacunación de esta persona (ordenadas por fecha) en una estructura de tipo ***tCSVData***, en que habrá una entrada (***tCSVEntry***) de tipo “APPOINTMENT” para cada cita de vacunación en el formato:

“date;time;CP;vaccine”

- **date**: Hace referencia a la fecha de la cita de vacunación en el formato dd/mm/yyyy (dd: día con dos dígitos, mm: mes con dos dígitos, yyyy: año con cuatro dígitos)
- **time**: Hace referencia a la hora de la cita de vacunación en el formato hh:mm (hh: hora con dos dígitos (00-23), mm: minutos con dos dígitos (00-59)).
- **CP**: Hace referencia al código postal del centro de salud.
- **vaccine**: Hace referencia al nombre de la vacuna.

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 existe ninguna persona con este documento de identidad.

Nota: En los fitcheros **csv.h** y **csv.c** tenéis los métodos necesarios para trabajar con las estructuras **tCSVData** y **tCSVEntry**. Para simplificar, podéis asumir que no habrá citas de vacunación para una persona en más de un centro de salud.

Ejercicio 3: Enlace con los stocks de vacunas [30%]

Finalmente, queremos enlazar las citas de vacunación con la información de stocks de vacunas de los distintos centros.

Se pide:

a) Implementa el método **api_checkAvailability** del fichero **api.c**, que dada una estructura de tipo **tApiData**, el código postal de un centro de salud, el nombre de una vacuna y una fecha, nos indica si existen vacunas en stock para esta vacuna. Hay que tener en cuenta que:

- **i)** En caso de que la vacuna solamente requiera una dosis, consideramos que existe stock si existe un número de dosis disponibles para la fecha dada superior o igual a 1.
- **ii)** En caso de que la vacuna requiera varias dosis, consideramos que hay stock si hay al menos una dosis de la vacuna en las fechas de cada dosis necesaria, contando por la fecha dada e incrementando según los días que deben pasar según la vacuna. Hay que tener en cuenta que para qué la segunda dosis esté disponible, debe haber un mínimo de dos dosis en la fecha de esta segunda dosis (la de la primera y la de esta segunda).
- **iii)** En caso de que el centro o la vacuna no existan, se asume que no existe disponibilidad.

b) Implementa el método **api_findAppointmentAvailability** del fichero **api.c**, que dada una estructura de tipo **tApiData**, el código postal de un centro de salud, el documento de identificación de una persona, y una fecha y hora, le asigne a esa persona las citas de vacunación necesarias para obtener la pauta completa. Hay que tener en cuenta las siguientes restricciones:

- **i)** La primera cita debe estar entre la fecha dada y los 6 días siguientes (total una semana).
- **ii)** Para que haya disponibilidad, el centro debe tener dosis disponibles para todas las dosis necesarias de al menos una vacuna. En caso de que haya más de una vacuna con dosis disponibles, se seleccionará una indistintamente.
- **iii)** Si no hay disponibilidad para ningún día, se le asignará el primer día de la semana siguiente en el que haya dosis de alguna vacuna, aunque no haya dosis en stock para todas las dosis de la pauta completa. Habrá que añadir igualmente todas las citas de vacunación necesarias. En caso de que no quede ninguna dosis de ninguna vacuna la siguiente semana, no se añade ninguna cita de vacunación, y se devuelve un error (**E_NO_VACCINES**)

iv) Las citas de vacunación se añadirán a la misma hora del parámetro de entrada, pero en los días correspondientes.

v) Una vez añadidas las citas de vacunación, será necesario actualizar el stock para descontar las vacunas correspondientes a las citas de vacunación añadidas. Tened en cuenta que en caso de que no hubiera suficiente dosis en stock, pueden quedar valores negativos. Para simplificar, no es necesario tener en cuenta actualizaciones retroactivas.

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 existe ninguna persona con este documento de identidad.
E_HEALTH_CENTER_NOT_FOUND	No existe ningún centro de vacunación con este código postal.
E_DUPLICATED_PERSON	Esta persona ya tiene citas de vacunación en este centro.
E_NO_VACCINES	No hay vacunas para atender una nueva cita de vacunación.

Nota: En los ficheros **date.h** y **date.c** tenéis el método **dateTime_addDay** que permite añadir un cierto número de días a un objeto de tipo **tDateTime**.