Prácticas de Programación
PRA1 - 20202

Fecha límite de entrega: **22 / 3 / 2021**

Estudiante

Apellidos: _____

Nombre: _____

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 **22 de marzo de 2021 a las 23:59**.

Se debe entregar un fichero en formato ZIP que contenga un directorio **‘UOC2020_2’** 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 ‘UOC2020_2’ (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.
- 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.

Aviso

Aprovechamos para recordar que está totalmente prohibido copiar en las PRs de la asignatura. Se entiende que puede haber un trabajo o comunicación entre los alumnos durante la realización de la actividad, pero la entrega de ésta tiene que ser individual y diferenciada del resto.

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.

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.

Nota: al conjunto de pruebas que se os dan con el enunciado se le suele llamar “test unitarios”.

Objetivos

- Compilación de un proyecto de código.
- Comprensión del proyecto a desarrollar.
- Familiarización con las funcionalidades y el uso del código proporcionado por los profesores.
- Recordatorio de la sintaxis, el estilo y el entorno de programación en C.
- Utilización de memoria dinámica.

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 el apartado «Introducción», concretamente las transparencias de síntesis TS2, TS3 y TS4.

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

La Unión Europea ha decidido crear un software para la gestión de la vacunación contra COVID 19. Esta aplicación deberá gestionar toda la información sobre las vacunas de las distintas farmacéuticas y para la correcta administración de estas, con el objetivo de tener al 70% de los adultos europeos vacunados contra el coronavirus el 21 de septiembre, día en el que finaliza formalmente el verano.

Analizando las necesidades de la nueva aplicación se han identificado los requisitos siguientes:

Se debe poder gestionar un conjunto de **países (country)**, almacenando su nombre (name) y si pertenece a la unión europea (es posible gestionar países que no pertenecen a la UE).

Se debe de poder gestionar las **vacunas (vaccine)** según tecnología (technology) (Adenovirus, SARS-CoV-2 Inactivo, peptídica o ARN), fase actual (Preclínica, fase I, fase I-II, fase II, fase II-III o fase III) y una lista de países que la tienen autorizada

Se debe poder gestionar un conjunto de **desarrolladores (developer)**, almacenando distintos parámetros como su nombre (name), país (country) y vacuna (vaccine).

En esta primera práctica se propone la implementación de los métodos para la gestión básica de las estructuras de datos asociadas al modelo que se propone: desarrolladores y vacunas.

Ejercicio 1: Preparación del entorno [30%]

Junto con el enunciado encontraréis un fichero ZIP con el código de la práctica. La estructura es:

- **UOCVaccine**
  - **UOCCovid19Vaccine**: Librería estática para gestionar el sistema de análisis de los agentes infecciosos.
    - **src**: Ficheros de código (*.c)
      - Los ejercicios que se proponen en la práctica deben completarse en los ficheros existentes en este directorio.
    - **include**: Ficheros de cabecera (*.h)
  - **UOCProgrammeEC**: Proyecto ejecutable que utiliza el sistema de análisis implementado en la librería estática, y ejecuta un conjunto de test automatizados.
    - **src**: Fichero main.c, punto de entrada.
    - **test**: Contiene la implementación de las pruebas que permiten comprobar las funcionalidades de la librería.
      - **src**: Ficheros de código (*.c)
      - **include**: Ficheros de cabecera (*.h)

Será necesario que se cree un nuevo espacio de trabajo (Workspace) en CodeLite llamado UOCProgrammeEC, que contenga dos proyectos (que también debéis crear). Uno de tipo ejecutable **UOCProgrammeEC** y el otro de tipo librería estática **UOCCovid19Vaccine**.

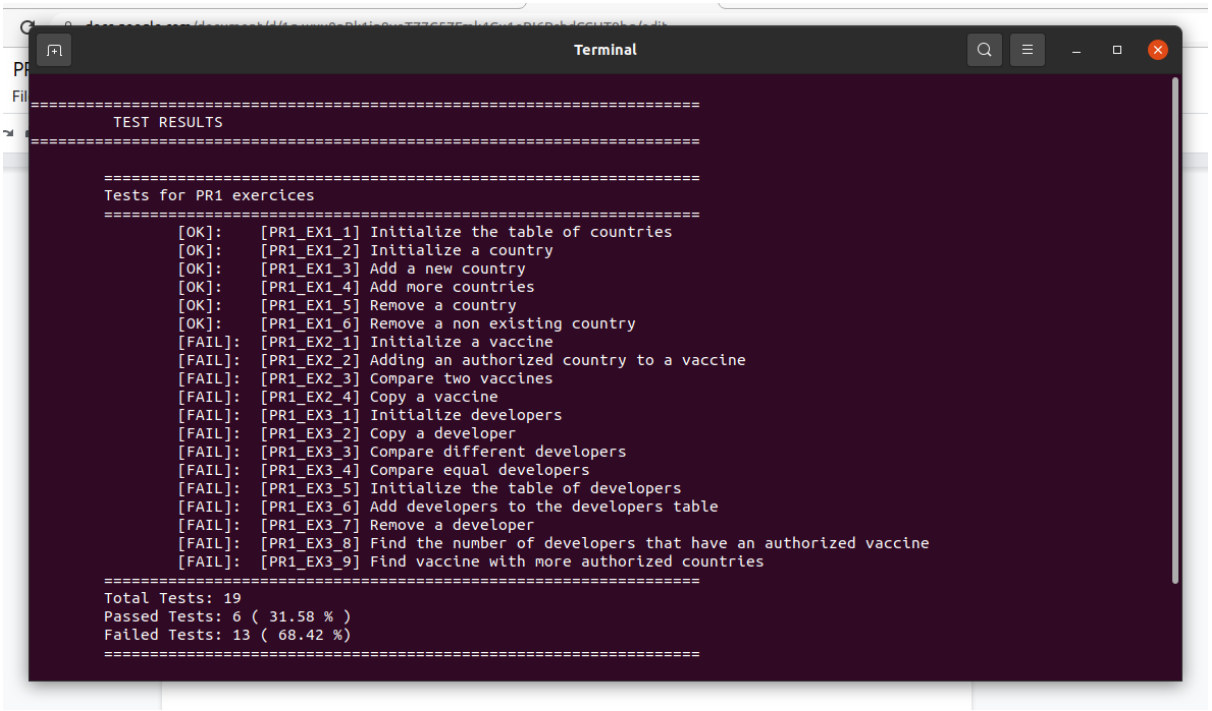
Para que el proyecto compile correctamente, se debe configurar:

- **UOCCovid19Vaccine**:
  - Se debe añadir el directorio “include” en la lista de directorios donde CodeLite va a buscar los ficheros de cabecera.

- **UOCProgrammeEC**:
  - Se debe añadir el directorio “include” y el directorio “test/include” en la lista de directorios donde CodeLite va a buscar los ficheros de cabecera.
  - Se debe añadir el directorio “include” de la librería **UOCCovid19Vaccine**.
  - Se debe indicar que este proyecto depende de la librería **UOCCovid19Vaccine**.

Nota: En los materiales de la asignatura encontraréis un apartado llamado “Modularidad en CodeLite” que contiene indicaciones para realizar este paso.

Una vez configurado correctamente el proyecto, al ejecutar el proyecto **UOCProgrammeEC** debería aparecer una salida como la que se muestra en la captura de pantalla siguiente. La salida muestra el conjunto de pruebas que se han ejecutado, y para cada conjunto de pruebas si ha pasado (OK) o ha fallado (FAIL), con una pequeña descripción de qué se está probando.



Comprobad que todas las pruebas para el ejercicio 1 de la práctica 1 **[PR1_EX1_XX]** pasan correctamente. Puede ser que alguna prueba de otros ejercicios esté como pasada, pero su estado puede cambiar a medida que se vaya avanzando en el proyecto.

Ejercicio 2: Gestión de las vacunas [30%]

Dentro del proyecto, en el fichero country.h encontraréis la definición de un tipo de dato para almacenar la información de un país (tCountry). Las funciones country_init, country_free, country_equals y country_cpy están implementadas y comentadas en el fichero country.c explicando puntos clave en la gestión de la memoria dinámica. Asimismo encontraréis el tipo tCountryTable que permite almacenar una tabla de países y que se comenta más adelante.

A lo largo de éste y el resto de ejercicios, en caso de que no se pueda reservar o liberar memoria mediante las funciones de la librería estándar de C, se devolverá un error con código ERR_MEMORY_ERROR en el método correspondiente – excepto en aquellos métodos que no hagan ninguna llamada a funciones de gestión de memoria dinámica. Ver definición de códigos de error en error.h dentro del proyecto del enunciado.

En caso de ejecución correcta del método propuesto devolveremos el código OK – excepto en las funciones donde en su enunciado se pide explícitamente el retorno de un valor particular o las funciones que devuelven void.

Utilizando como referencia el código existente de country.c, dada la definición de **tVaccine** en el fichero vaccine.h implementa los métodos siguientes en el fichero vaccine.c:

1. tError vaccine_init(): Inicializa un dato de tipo tVaccine, a partir de los valores recibidos como parámetro durante su llamada, almacenando su nombre, tecnología y fase actual. Además se deberá inicializar una tabla tCountryTable vacía, que servirá para almacenar los países que la han autorizado .
2. void vaccine_free(): Elimina un dato de tipo tVaccine, liberando la memoria utilizada por éste y todos sus datos asociados.
3. bool vaccine_equals(). Compara dos vacunas. Se considera que dos vacunas son iguales si tienen el mismo nombre.
4. tError vaccine_add_authorized_country(). Añade a una vacuna un país que autoriza el uso de la vacuna.
5. tError vaccine_copy(). Realiza una copia de una vacuna.

Una vez implementados todos los métodos especificados en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 2 **[PR1_EX2_XX]**. Los principales conceptos que deben quedar claros después del ejercicio son:

- Utilización de las funciones malloc, realloc y free de la librería estándar de C
- Uso básico de punteros y significado del operador dirección (&)

Ejercicio 3: Gestión de los Desarrolladores [40%]

Al igual que hemos realizado con las vacunas, tenemos definidas tres funciones básicas dentro del fichero developer.h para la gestión de un desarrollador (tDeveloper) que han desarrollado vacunas candidatas que están en fase 3 y que ya están aprobadas por la unión europea.

Dada la definición de **tDeveloper** del fichero developer.h implementa los métodos siguientes en el fichero developer.c:

1. tError developer_init(): Inicializa un dato de tipo tDeveloper, a partir de los valores recibidos como parámetro durante su llamada.
2. void developer_free(): Elimina un dato de tipo tDeveloper, liberando la memoria utilizada por éste.
3. bool developer_equals(). Compara dos desarrolladores. Se considera que dos desarrolladores son iguales si tienen el mismo nombre.
4. tError developer_cpy(). Realizar una copia de un desarrollador.

Si os fijáis en las descripciones de las pruebas para el ejercicio 1, veréis que muchas de ellas se refieren a la gestión de una estructura que almacena países. En concreto, en el fichero country.h encontraréis la definición de una tabla de países tCountryTable.

En el fichero country.c encontraréis implementados los métodos de inicialización y eliminación para la estructura tCountryTable junto a las operaciones básicas para añadir, buscar y eliminar elementos a la tabla de países. Estas funciones están comentadas para explicar los puntos más importantes sobre la gestión de la memoria dinámica.

En programación a menudo nos encontramos en la situación de tener que adaptar un código que hace algo similar a lo que tenemos que hacer, a fin de resolver nuestro problema. El objetivo de este ejercicio es entender la implementación dada para la gestión de las tablas de países (tCountryTable) y utilizarla como referencia para implementar la gestión de la tabla de desarrolladores (tDeveloperTable).

Dada la definición de **tDeveloperTable** en el fichero developer.h implementa los métodos siguientes en el fichero developer.c:

5. void developerTable_init(): Inicializa una tabla de desarrolladores.
6. int developerTable_size(): Devuelve el tamaño de la tabla de desarrolladores.
7. developerTable_free(): Libera la memoria que se haya reservado dinámicamente para guardar la tabla de desarrolladores.
8. tError developerTable_add(): Añade un desarrollador a la tabla. Si este ya se encuentra en la tabla, se devolverá el valor de error ERR_DUPLICATED.
9. tError developerTable_remove(): Elimina un desarrollador de la tabla. En caso de que se intente eliminar un desarrollador que no existe, se devolverá el valor de error ERR_NOT_FOUND.
10. int developerTable_num_authorized(): Devuelve el número de desarrolladores que tienen una vacuna autorizada en al menos un país
11 int developerTable_most_popular(): Devuelve la posición que ocupa en la tabla el desarrollador que tiene una vacuna con más países que la autorizan. En caso de empate se seleccionará el desarrollador que se encuentre primero en la tabla. Si la tabla está vacía, se devolverá el valor de error ERR_EMPTY_LIST

Además tenéis implementadas algunas funciones que os pueden ser de utilidad para realizar este ejercicio:

12. int developerTable_find(): dado un desarrollador encuentra la posición que este ocupa en la tabla de desarrolladores. Se devolverá el valor de error ERR_NOT_FOUND si este no se encuentra en la tabla.
13. int developerTable_size(): devuelve el número de desarrolladores que contiene la tabla.
14. developerTable_print() y developer_print(): imprime por pantalla información básica de la tabla de desarrolladores y de un desarrollador, respectivamente.

Una vez implementados todos los métodos especificados en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 3 **[PR1_EX3_XX]**. Los principales conceptos que deben quedar claros después del ejercicio son:

- Definición y gestión de tablas con memoria dinámica.