Prácticas de Programación
PR2 - 20202

Fecha límite de entrega: **19/04/2021**

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 **19 de abril de 2021 a las 23:59.**

Se entregará 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 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 en la máquina virtual de FP** 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 en la máquina virtual de FP**. 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.
- 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

- Trabajar con Tipos Abstractos de Datos (TAD).
- Implementar algoritmos recursivos.

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 los apartados «Recursividad I», «Recursividad II» y «TAD en memoria dinámica».

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).
- Se debe poder gestionar una cola de **pacientes** (patient), almacenando distintos parámetros como su nombre (name), identificador, la vacuna (vaccine) que se le ha administrado, y el número de dosis administradas (0 si aún no está vacunado).

En esta segunda práctica se propone la implementación de una **cola** de pacientes y la extracción de algunos datos estadísticos usando **recursividad**.

Ejercicio 1: Creación de una cola [35%]

Junto con el enunciado tenéis el código de la práctica anterior modificado, en el que hemos añadido los archivos *patient.h* y *patient.c* y además esta vez se incluye el workspace. Estos archivos contienen los tipos de datos y las funciones para la gestión de los pacientes.

Cada paciente **(tPatient)** tiene asociada una vacuna **(tVaccine).** Los pacientes se guardarán en una cola de pacientes **(tPatientQueue),** donde los nodos serán de tipo **tPatientQueueNode**.

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 la función correspondiente – excepto en aquellos funciones 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 de la función propuesta 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.

Se pide:

1. Dada la definición de **tPatientQueue** en el fichero patient.h, implementa los funciones siguientes en el fichero patient.c:
    a. **patientQueue_create:** Inicializa una estructura tPatientQueue como una cola vacía. Devuelve true si la ejecución es correcta, false en caso contrario.
    b. **patientQueue_empty:** Indica si la cola que se le pasa está vacía (true) o por el contrario, contiene elementos (false).
    c. **patientQueue_enqueue**: Añade un nuevo elemento a la cola y devuelve un código con el resultado de la ejecución.
    d. **patientQueue_free:** Elimina todos los elementos de la cola.

2. Dada la definición modificada de **tCountry** del archivo *country.h,* donde se ha añadido una tabla de vacunas autorizadas y la cola de pacientes (*patients*):
    a. Se debe modificar la implementación de la función **country_init** para que también inicialice la cola de pacientes.
    b. Implementa la función **country_addPatient** para que añada un nuevo paciente al país. La función devuelve un código con el resultado de la ejecución.
    c. Implementa la función **countryTable_addPatient** para que añada un nuevo paciente a la tabla que contenga el nombre del país. La función devuelve un código con el resultado de la ejecución.

Nota: Disponéis de las funciones **patient_init** y **patientQueue_duplicate,** que os pueden ser de ayuda para realizar este ejercicio, pero no es necesario utilizarlos.

Una vez implementados todas las funciones especificadas en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 1 [**PR2_EX1_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Definición de una cola
- funciones para crear y añadir elementos a una cola

Ejercicio 2: Operaciones de acceso a una cola [35%]

En el ejercicio anterior hemos trabajado con las funciones para crear una cola y con funciones que nos permiten añadir información. En este ejercicio trabajaremos con funciones que permiten obtener información de la cola y destruirla. Recuerda que las colas son un tipo de datos que sólo permiten acceder al primer elemento que se ha añadido, y que los nuevos elementos van a parar al final de la cola.

Se pide:

1. Dada la definición de **tPatient** y **tPatientQueue** en el fichero *patient.h,* implementa las funciones siguientes en el archivo patient.c:

    a. **patient_inoculate_vaccine:** inocula una dosis de una vacuna a un paciente, si este ya tenía una vacuna y se le intenta inyectar otra diferente, devolverá un error ERR_INVALID_VACCINE.
    b. **patientQueue_head:** Nos devuelve por referencia el elemento que está al frente de la cola. Este elemento no se eliminará de la cola. En caso de que la cola esté vacía, nos devolverá un error ERR_NOT_FOUND.
    c. **patientQueue_dequeue**: Elimina el elemento del frente de la cola. La función retorna NULL si la cola está vacía. Si la cola tiene algún elemento elimina el elemento del frente de la cola y lo devuelve.
    d. **patientQueue_free:** Elimina todos los elementos de la cola. Si la cola está vacía, no hay que hacer nada. Devolverá *void*.

2. Dada la definición modificada de **tCountry** al archivo *country.h,* en que se ha añadido la cola de pacientes (patients):

    a. Modifica la implementación de la función **country_free** para que al eliminar el país se eliminen sus pacientes.
    b. Implementa la función **country_getMostUsedVaccineTechnology** para que, dado un país (tCountry), devuelva la tecnología de vacuna (tVaccineTec) que más pacientes de ese país han usado. Si el país no tiene pacientes devuelve NONE. Los pacientes que no están vacunados aún no cuentan.

Nota: Tenéis un ejemplo de recorrido de colas en la función **patientQueue_compareIterative**.

Disponéis de la función **patientQueue_duplicate,** que nos hace un duplicado de una cola en el mismo orden. Esta función permite crear una copia de una cola, evitando eliminar los elementos de la cola inicial y también evitando perder el orden al recorrer todos los elementos mediante enqueue / dequeue, trabajando en una copia.

Utiliza las funciones **patientQueue_head** y **patientQueue_dequeue** para desencolar todos los elementos de uno en uno y obtener las tecnologías de las vacunas.

Utiliza un vector de tantas posiciones como tipos de tecnologías conocidas para almacenar las veces que aparece cada tipo entre los pacientes del país.

Una vez implementados todos las funciones especificadas en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 2 [**PR2_EX2_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- funciones para acceder y eliminar elementos de una cola.
- Cómo duplicar una cola para evitar perder los datos iniciales.

Ejercicio 3: Recursividad [30%]

Cuando se trabaja con TADs, una práctica habitual es la utilización de funciones recursivos para resolver problemas. En el archivo *patient.c* se muestra la función **patientQueue_compareIterative** que es un ejemplo de una implementación iterativa para comparar dos colas. Esta función no hace copias de las colas y, por tanto, al acabar la ejecución las colas estarán vacías.

Se pide implementar de **forma recursiva** las siguientes funcionalidades a *country.c* y patient.c:

1. **patientQueue_compareRecursive:** función que hace exactamente lo mismo que la función ya implementado dentro de *patient.c* **patientQueue_compareIterative,** comparar si dos colas son iguales, pero en este caso os pedimos que se haga de forma recursiva. La función no hará una copia de las colas. Para que estas no queden vacías se ha creado también la función **patientQueue_compare**, que os damos implementada, que hace la copia de las colas y llama a **patientQueue_compareRecursive** para compararlas.

2. **country_getPatientsPerVaccine:** dado un país (tCountry) y una vacuna (tVaccine), cuenta cuántos pacientes ya vacunados tiene el país con esa vacuna mediante la función **patientQueue_getPatientsPerVaccineRecursive.** En caso de que no haya ninguno, devolverá cero.

3. **country_getPatientsPerVaccineTechnology:** dado un país (tCountry) y una tecnología de vacuna (tVaccineTec), cuenta cuántos pacientes del país se han vacunado ya con esta tecnología mediante la función **patientQueue_getPatientsPerVaccineTechnologyRecursive.** En caso de que no haya ninguno, devolverá cero.

Una vez implementados todas las funciones especificadas en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 3 [**PR2_EX3_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Cómo definir una función recursiva