Prácticas de Programación
PR3 - 20202

Fecha límite de entrega: **17/05/2021**

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 **17 de mayo 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 «TAD en memoria dinámica», «Métodos de búsqueda» y «Métodos de ordenación».

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

Actualmente ya hay cuatro vacunas aprobadas por la Agencia Europea de Medicamentos (EMA por sus siglas en inglés): Pfizer, Moderna, Astrazeneca y Janssen, esta última de una sola dosis.

Para facilitar el proceso de vacunación se va a aumentar la funcionalidad de la aplicación UOCCovid19Vaccine añadiendo lotes de vacunas (**VaccinationBatch**).

Además se ha decidido indicar en cada paciente (**patient**) a qué grupo poblacional pertenece. De más a menos prioridad son los siguientes:

1. Personal sanitario en contacto con pacientes
2. Personas de 80 años o más de residencias.
3. Personas adultas de 65 a 79 años
4. Personas con comorbilidades
5. Personal esencial
6. Mayores de 55 años
7. Vacunas para el resto de población.

Se debe tener en cuenta que:

- La vacuna “AZD1222” del desarrollador Oxford/Astrazeneca no puede ser implementada en los grupos 2 y 3
- La vacuna “JNJ-78436735” del desarrollador Janssen es de una sola dosis.

En esta tercera práctica se propone la implementación de una **lista** de lotes de vacunas disponibles de cada país que se irán administrando a los pacientes según su grupo prioritario.

Ejercicio 1: Creación de una lista de vacunas [35%]

Junto con el enunciado tenéis el código de la práctica anterior modificado, en el que hemos añadido los archivos *vaccinationBatch.h* y *vaccinationBatch.c.* Estos archivos contienen los tipos de datos y las funciones para la gestión de los lotes de vacunas.

Cada lote **(tVaccinationBatch)** tiene asociada una vacuna **(tVaccine),** un número de lote y un número de vacunas. Los lotes de vacunas se guardarán en una **lista** (**tVaccinationBatchList)** donde los nodos serán de tipo **tVaccinationBatchListNode.**

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:

Dada la definición de **tVaccinationBatchListNode** en el fichero *vaccinationBatch.h,* implementar las funciones siguientes en el fichero *vaccinationBatch.c:*

1. **vaccinationBatchList_create()**: inicializa una estructura **tVaccinationBatchList** como una lista vacía.

2. **vaccineBatchList_insert()**: Inserta un elemento en la posición indicada por el índice en la lista (empezando por el cero). Si el índice es mayor que el tamaño de la lista, devolverá ERR_INVALID_INDEX.

3. **vaccineBatchList_delete()**: elimina el elemento de la posición indicada por el índice de la lista (empezando por el cero). Si el índice es mayor que el tamaño de la lista, devolverá ERR_INVALID_INDEX, en cambio si la lista está vacía devolverá ERR_EMPTY_LIST.

4. **vaccineBatchList_get()**: devuelve el nodo de la posición indicada por el índice de la lista. En caso de que el índice sea mayor que el tamaño de la lista o esté vacía devolverá NULL.

5. **vaccinationBatchList_empty()**: indica si la lista que se le pasa está vacía (true) o, por el contrario, contiene elementos (false).

6. **vaccinationBatchList_free()**: elimina todos los elementos de la lista.

Una vez implementados todas las funciones especificadas en el ejercicio deberían pasar correctamente todos los test referentes al ejercicio 1 [**PR3_EX1_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Definición de una **lista.**

Ejercicio 2: Gestión de las vacunas de un país [35%]

En el ejercicio anterior hemos trabajado con las funciones para crear una lista de lotes de vacunas con funciones que nos permiten añadir información. En este ejercicio trabajaremos con un país (**tCountry**) que ahora tendrá almacenada una lista de lotes de vacunas (**tVaccinationBatchList**) que deberá administrar en la cola de pacientes (**tPatientQueue**) a la espera de vacunarse.

Se pide:

1. Dada la nueva definición de **tCountry** con una lista de lotes de vacunas disponibles **tVaccinationBatchList** en el fichero *country.h,* implementa las siguientes funciones en el archivo country.c. Nota: para hacer los apartados a y b hay que llamar a las funciones de la segunda parte de este ejercicio.

    a. **country_inoculate_first_vaccine:** inocula todas las dosis disponibles de cada lote de vacunas a la cola de pacientes a los que no se les haya administrado ninguna vacuna. Teniendo en cuenta que los grupos a los que puede asignarse cada vacuna. Los lotes vacíos no se han de eliminar de la lista. Se aplica la vacunación en el orden que hay en la cola de pacientes.
    b. **country_inoculate_second_vaccine:** inocula todas las dosis disponibles de la lista de lotes de vacunas a la cola de pacientes a los pacientes que no se les haya administrado la segunda vacuna. Teniendo en cuenta el número de dosis de cada vacuna y que no se pueden administrar vacunas diferentes a la de la primera dosis. Los lotes vacíos no se han de eliminar de la lista. Se aplica la vacunación en el orden que hay en la cola de pacientes.
    c. **country_percentage_vaccinated.** Devuelve el tanto por ciento de la cola de pacientes del país que han sido vacunados con todas las dosis (hay vacunas de dos dosis y vacunas de una dosis)

2. Dada la nueva definición de **tPatient** con un grupo de población en el fichero patient.h, implementa las funciones siguientes en el archivo vaccinationBatch.c:

    a. **vaccineBatchList_inoculate_first_vaccine**: inocula la primera vacuna de la lista de lotes de vacunas llamando a la función vaccineBatchList_inoculate.
    b. **vaccineBatchList_inoculate_second_vaccine**: inocula la segunda vacuna de la lista de lotes de vacunas llamando a la función vaccineBatchList_inoculate.
    c. vaccineBatchList_inoculate: Función que recorre la lista de lotes que es apropiada para inocularsela al paciente.

3. Dada la nueva definición de **tPatient** con un grupo de población en el fichero patient.h, implementa las siguientes funciones en el archivo patient.c:

    a. patient_isSuitableForVaccine: dado un paciente y una vacuna devuelve cierto si la vacuna es adecuada para el paciente, y falso en caso contrario.

Una vez implementadas todas las funciones especificadas en el ejercicio, deberían pasar correctamente todos los test referentes al ejercicio 2 [**PR3_EX2_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Cálculos matemáticos básicos utilizando un TAD

Ejercicio 3: Búsqueda y ordenación [30%]

En el trabajo con TAD es habitual realizar recorridos, búsquedas o ordenaciones. En este apartado se pide que implementéis las siguientes funciones:

1. Implementa la función siguiente en el archivo patient.c:
    a. **patientQueue_countPatients_vaccinationBatch():** dado un identificador de lote de una vacuna, devolverá el número de pacientes que han sido vacunados con ese lote de vacuna. La función debe ser **secuencial**.

2. Implementa las funciones siguientes en el archivo vaccinationBatch.c:
    a. **vaccineBatchList_quickSortRecursive**(): dada una lista desordenada de lotes de vacunas, los ordena según identificador del lote (lotID) y de manera descendente usando el nombre. Se debe implementar el algoritmo **Quicksort** (**ordenación rápida**).
    b. **vaccineBatchList_quicksort**(): función que llama a la función recursiva vaccineBatchList_quickSortRecursive**.**

Tenéis implementada las funciones **vaccineBatchList_swap** y **vaccineBatchList_getlotID** que os pueden ser de utilidad para realizar este ejercicio.

Una vez implementadas todas las funciones especificadas en el ejercicio deberían pasar correctamente todos los test referentes al ejercicio 3 [**PR3_EX3_XX**]. Los principales conceptos que deben quedar claros después del ejercicio son:

- Búsquedas dentro de una cola
- Ordenación de una lista con quicksort.