Prácticas de Programación
PR2 - 20211

Fecha límite de entrega: **14 / 11 / 2021**

Formato y fecha de entrega

La práctica debe entregarse antes del día **14 de noviembre 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 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.

Enunciado

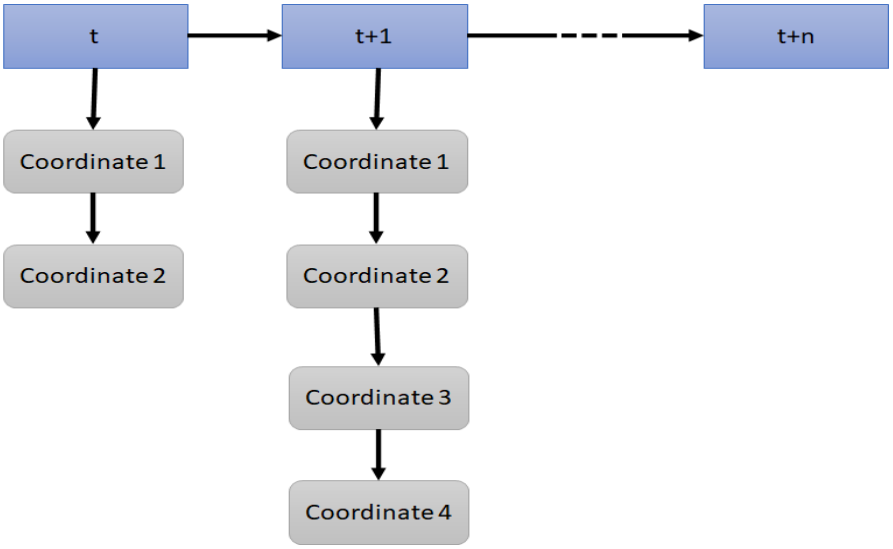
El punto de partida para esta práctica es el resultado de la PR1. El código facilitado con el enunciado incorpora todos los ficheros necesarios de las actividades anteriores, por lo que veréis que todas las pruebas de la PR1 pasan correctamente.

En esta práctica vamos a implementar la estructura de datos planteada en la PEC3 y los métodos relacionados, para poder analizar temporalmente los contactos estrechos:

```
type
      tCoordinateNode = record
            coordinate : tCoordinate;
            numPersons : integer;
            next: pointer to tCoordinateNode;
      end record

      tDateTimeNode = record
            dateTime : tDateTime;
            coordinatesList : pointer to tCoordinateNode;
            next: pointer to tDateTimeNode;
      end record
end type
```

Que guarda para cada momento temporal **t** la lista de coordenadas que contienen personas.



En los distintos ejercicios se irá implementando las estructuras y métodos necesarios.

Entorno de pruebas

Para facilitar la verificación de vuestro código y su estructura antes de la entrega, hemos puesto a vuestra disposición un mecanismo de verificación. Se trata de un sistema basado en Linux que verifica vuestra entrega y os notifica los resultados.

Para utilizarlo:

1. Preparad el **fichero ZIP** siguiendo las instrucciones de entrega.
    a. La estructura es la misma que el código facilitado con el enunciado.
    b. El **README.txt** debe contener vuestros datos.
    c. No debe incluir las carpetas **bin**, **lib**, ni otras carpetas resultantes de la compilación.
2. Enviad un correo desde vuestra **cuenta institucional de la UOC** con las siguientes características:
    a. **Dirección**: pelp.inbox@gmail.com
    b. **Asunto**: 8eec2ae7-e959-47dd-8caa-b666b62dc076
    c. **Contenido**: Adjuntar el fichero ZIP
3. En **unas horas**, vais a recibir un correo con el resultado de la ejecución de vuestro código y el resultado de las pruebas.
    a. No es un sistema de entrega, tenéis que entregar igualmente vuestra práctica en el **apartado de EC del aula de teoría**.
    b. No es un sistema de compilación. Esta herramienta está pensada para que podáis validar que el código que os funciona en local va a funcionar en un entorno de prueba.
    c. **Hay un límite** de 2 entregas por día y un máximo de 10 entregas en total.
    d. No esperéis al último día, ya que quizás no tengáis los resultados antes de la fecha límite de entrega.
    e. El sistema no acepta entregas pasada la fecha límite de la actividad.

Nota: Este sistema está en fase de pruebas, si encontráis algún problema indicadlo a los profesores para que podamos corregir cualquier incidencia.

Ejercicio 1: Integración en los datos de la API [10%]

En el código proporcionado encontraréis los ficheros ***datetime_node.h*** y ***datetime_node.c*** que contienen la implementación de los tipos de datos **tCoordinateNode** y **tDateTimeNode,** y las cabeceras de los métodos que implementaremos para manejarlos. Se pide:

a) Modifica la definición de los datos de la aplicación **tApiData** en el fichero **api.h** para que contengan la lista de nodos **tDateTimeNode**.
b) Modifica el método **api_initData** del fichero **api.c** para que tenga en cuenta durante la inicialización el cambio del apartado anterior.
c) Implementa el método **api_getTemporalData** del fichero **api.c**, para que dada una estructura de datos **tApiData**, nos retorne la lista de coordenadas temporales de la aplicación, creada en el primer punto del ejercicio.

Ejercicio 2: Inserción de datos en las listas [40%]

En la práctica anterior implementamos el método **api_addDataEntry** para añadir nuevos datos a la aplicación. Este método recibía un dato de tipo “PERSON” o “GEOLOCATION”, y lo añadía a la aplicación si se cumplían una serie de condiciones.
Queremos que cuando se añada un nuevo dato de posicionamiento (“GEOLOCATION”), este quede reflejado en la lista de posiciones que hemos introducido en el ejercicio anterior. Se pide:

a) Implementa el método ***coordinateNode_insert*** del fichero ***datetime_node.c*** que dada una lista de coordenadas **tCoordinateNode** por referencia, y un nuevo dato temporal **tTimePosition**, actualice la lista para reflejar este nuevo dato. La actualización se debe realizar del siguiente modo:
  - **i)** Si existe un nodo con la misma coordenada, se incrementará el valor numPersons de ese nodo.
  - **ii)** Si no existe ningún nodo para la coordenada, se añadirá un nuevo nodo al final de la lista para la nueva coordenada, con un valor de número de personas (numPersons) igual a uno.

b) Implementa el método ***dateTimeNode_insert*** del fichero ***datetime_node.c*** que dada una lista temporal **tDateTimeNode** por referencia, y un nuevo dato temporal **tTimePosition**, actualice la lista para reflejar este nuevo dato. La actualización se debe realizar del siguiente modo:
  - **i)** Si existe un nodo para el mismo momento temporal (fecha y hora), se añadirá el nuevo dato en el nodo existente utilizando el método ***coordinateNode_insert*** anterior.
  - **ii)** Si no existe ningún nodo para este momento temporal, se añadirá un nuevo nodo, **de forma que la lista quede ordenada temporalmente (orden ascendente)**.

c) Modifica la implementación del método ***api_addDataEntry*** del fichero **api.c**, para que cuando se añade un nuevo dato, este se refleje en la lista de posiciones **tDateTimeNode** que hemos añadido en el ejercicio anterior.

Nota: Podéis utilizar la función ***coordinate_equals*** definida en el fichero **contact.h** y la función ***dateTime_cmp*** definida en el fichero **date.h** para realizar este ejercicio.
Nota: Al igual que cuando pasamos un entero (int) por referencia, este se convierte en un puntero a entero (int*), cuando pasamos un puntero por referencia (int*), este se convierte a un puntero a un puntero (int**). Si os confunde el doble puntero, podéis definir un tipo intermedio **tDateTimeNodeList** de tipo **tDateTimeNode***, con lo que evitaréis el doble puntero. Lo mismo aplica a **tCoordinateNode**.

Ejercicio 3: Cálculo sobre listas [20%]

En el ejercicio 2 de la PEC3 se pedía la implementación en lenguaje algorítmico de las funciones recursivas ***numPeopleCoordinates*** y ***numPeopleTime***. En este ejercicio se pide hacer su implementación en lenguaje C:

a) Implementa la función ***numPeopleCoordinates*** del fichero ***datetime_node.c*** para que calcule el número de personas totales en una lista de coordenadas **tCoordinateNode**. Utiliza una implementación **recursiva**.

b) Implementa la función ***numPeopleTime*** del fichero ***datetime_node.c*** para que dada una lista **tDateTimeNode** calcule el número de personas totales entre dos momentos temporales. Utiliza una implementación **recursiva**.

c) Implementa la función ***api_numPersons*** del fichero ***api.c*** para que dada una estructura **tApiData** con los datos de la aplicación, calcule el número de personas totales en un rango de fechas.

Ejercicio 4: Eliminación de datos [30%]

Finalmente, queremos que los nuevos datos añadidos en esta práctica se eliminen al eliminar los datos de la aplicación. Para ello:

a) Implementa la función ***coordinateNode_insert*** del fichero ***datetime_node.c*** para que dada una lista de nodos **tCoordinateNode,** elimine todos los datos de la lista. Utiliza una implementación **recursiva**.

b) Implementa la función ***dateTimeNode_free*** del fichero ***datetime_node.c*** para que dada una lista de nodos **tDateTimeNode**, elimine todos los datos de la lista. Utiliza una implementación **recursiva**.

c) Modifica el método ***api_freeData*** del fichero **api.c** para que al eliminar los datos de la aplicación, también se eliminen los datos de la lista añadida en el primer ejercicio.