Prácticas de Programación
PR2 - 20241

Fecha límite de entrega: **02 / 12/ 2024**

Formato y fecha de entrega

Hay que entregar la práctica antes del día **2 de diciembre 2024** a las 23:59.

Se debe entregar un archivo en formato **ZIP**, que contenga una carpeta **UOC20241** con el directorio principal de vuestro proyecto, siguiendo la estructura de carpetas y nombres de archivos especificados en el enunciado de la práctica. No debe contener ningún archivo 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.
- Todos los ficheros deben estar dentro de las carpetas correctas (src, test, …).
- Si se ha utilizado el entorno CodeLite: los ficheros .workspace y .project que definen el espacio de trabajo y los proyectos de Codelite.

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 período 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). No es necesario pasar todos los tests pero por lo menos debe mostrarse el resultado de estos por pantalla.
    - Que se respeten los **criterios de estilo y** que el código esté **debidamente comentado**. Se valorará especialmente el uso de comentarios en inglés.
    - Que las **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.

En esta práctica sólo se podrán modificar los archivos ***invoice.c***, ***invoice.h***, ***tax_office.c***, ***tax_office.h***, ***api.c*** y ***api.h***, que serán los archivos a subir a la herramienta DSLab.

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

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 se utiliza 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.

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/

DSLab

Siguiendo con la misma dinámica que en la práctica anterior, se usará la herramienta DSLab (https://sd.uoc.edu/dslab/). Esta herramienta también se utiliza en otras asignaturas y tiene como objetivo:

- Proporcionar un entorno común en el que evaluar los ejercicios de codificación.

Os aconsejamos realizar envíos periódicos a la herramienta de los diferentes ejercicios de código, ya que os permitirá detectar posibles errores antes de la entrega final. **Tened presente que es la herramienta utilizada como base para corregir vuestros códigos, y que no se corregirá ningún código en otro entorno o máquina.** Así pues, si vuestro código no funciona en la herramienta DSLab, se considerará que no funciona, aunque lo haga en vuestro ordenador.

En todo caso, **hay que tener presente que las entregas finales deben seguir haciéndose en el apartado correspondiente del aula, tal como indica el enunciado**. Esta herramienta es una ayuda adicional que ponemos a vuestra disposición, y en ningún caso es obligatorio su utilización.

La información básica que os será de utilidad al utilizar DSLab:

- DSLab considera la entrega correcta únicamente si esta pasa todos los tests.
- Se muestra un resumen rápido del número de tests pasados. Por norma general no será necesario pasarlos todos para aprobar la entrega.
- En los detalles se muestra:
    - El detalle de los tests pasados y de los que han fallado.
    - Es posible descargar un fichero con el texto que el programa muestra por pantalla (salida estándar). Se ha incluido el uso de **valgrind** en la salida estándar de manera que en este apartado podréis ver el informe sobre la **gestión de la memoria**. Es recomendable revisar esta parte para asegurarse de que se hace un uso correcto de los punteros y de la memoria.
- Existe también un log de ejecución que guarda la evolución de la ejecución del programa. Si debido a una codificación incorrecta el programa falla y no es capaz de mostrar el resultado de los tests, se debe revisar este log para determinar en qué punto se interrumpió la ejecución.

Nota: aunque DSLab es un sistema robusto y utilizado en varias asignaturas de la UOC, para esta asignatura en concreto está en fase de pruebas. Si encontráis algún problema indicadlo a los profesores para que podamos corregir cualquier incidencia.

Enunciado

El punto de partida de esta práctica es la solución oficial de la PR1. El código base del enunciado incluye todos los archivos necesarios de las actividades anteriores, con los que veréis que todos los tests de la PR1 pasan correctamente.

En esta práctica utilizaremos la estructura de datos planteada en la PR1 con algunas modificaciones y nuevas estructuras con sus funciones relacionadas para poder gestionar los datos de los inquilinos, sus contratos y las oficinas de impuestos

```
type

      tTenant = record
            start_date : tDate;
            end_date : tDate;
            tenant_id : string;
            name : string;
            rent : float;
            age : integer;
            postal_code: string;
            cadastral_ref : string;
      end record

      tTenantData = record
            elems : pointer to array of tTenant;
            count : integer;
      end record

      tRentInvoice = record
            rent : float;
            cadastral_ref : string;
      end record

      tRentInvoiceNode = record
            elem : tRentInvoice;
            next : pointer to tRentInvoiceNode;
      end record

      tRentInvoiceMonthly = record
            month : tDate;
            first : pointer to tRentInvoiceNode;
            count : integer;
            next : pointer to tRentInvoiceMonthly;
      end record

      tRentInvoiceData = record
            first : pointer to tRentInvoiceMonthly;
```

```
            count : integer;
      end record

      tTaxOffice = record
            office_code : string;
            rentInvoices : tRentInvoiceData;
      end record

      tTaxOfficeNode = record
            elem : tTaxOffice;
            next : pointer to tTaxOfficeNode;
      end record

      tTaxOfficeList = record
            first : pointer to tTaxOfficeNode;
            count : integer;
      end record

end type
```

Ejercicio 1: Gestión de las facturas de un alquiler [50%]

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

a) El método ***invoiceList_init*** que dada una estructura de tipo **tRentInvoiceData**, la inicialice correctamente. Inicialmente, no habrá facturas para ningún mes.

b) El método ***invoiceList_update*** que dada una estructura de tipo **tRentInvoiceData**, de unas fechas de inicio y fin **tDate**, el catastro y la renta, cree las facturas mensuales de un propietario o las actualice si hay una actualización de la renta. Se debe tener en cuenta:

i) Si la lista de facturas está vacía, las facturas se generan automáticamente para los meses entre fecha de inicio y fin del primer contrato.

ii) Si se sube un contrato con la renta actualizada y ya existen facturas generadas, se deberá modificar la renta de las entradas posteriores a la fecha de inicio para reflejar el cambio.

iii) Todos los elementos están ordenados por la fecha, y entre dos fechas sólo hay un mes de diferencia.

iv) Dentro de una fecha, las facturas deben estar ordenadas por la referencia catastral.

Nota: Al final del ejercicio tenéis tres ejemplos de la evolución de las facturas generadas del propietario a partir de un estado inicial y varias llamadas al método ***invoiceList_update*** con una fecha de inicio, fecha de fin, referencia catastral y la renta. En los ficheros **date.h** y **date.c** encontraréis el método ***date_addMonth***, que os permite añadir o restar meses a una fecha dada. También encontraréis el método ***date_cmp*** que permite comparar dos fechas.

Nota: Para este ejercicio es muy recomendable aplicar la metodología del diseño descendente, definiendo todos los métodos adicionales que consideréis pero sin modificar los tipos de datos que os hemos proporcionado.

c) El método ***getInvoiceMonthly*** que dada una estructura de tipo **tRentInvoiceData**, una fecha **tDate**, nos retorna el total facturado de esa fecha. Si no hay generada ninguna factura para esa fecha, se devolverá 0

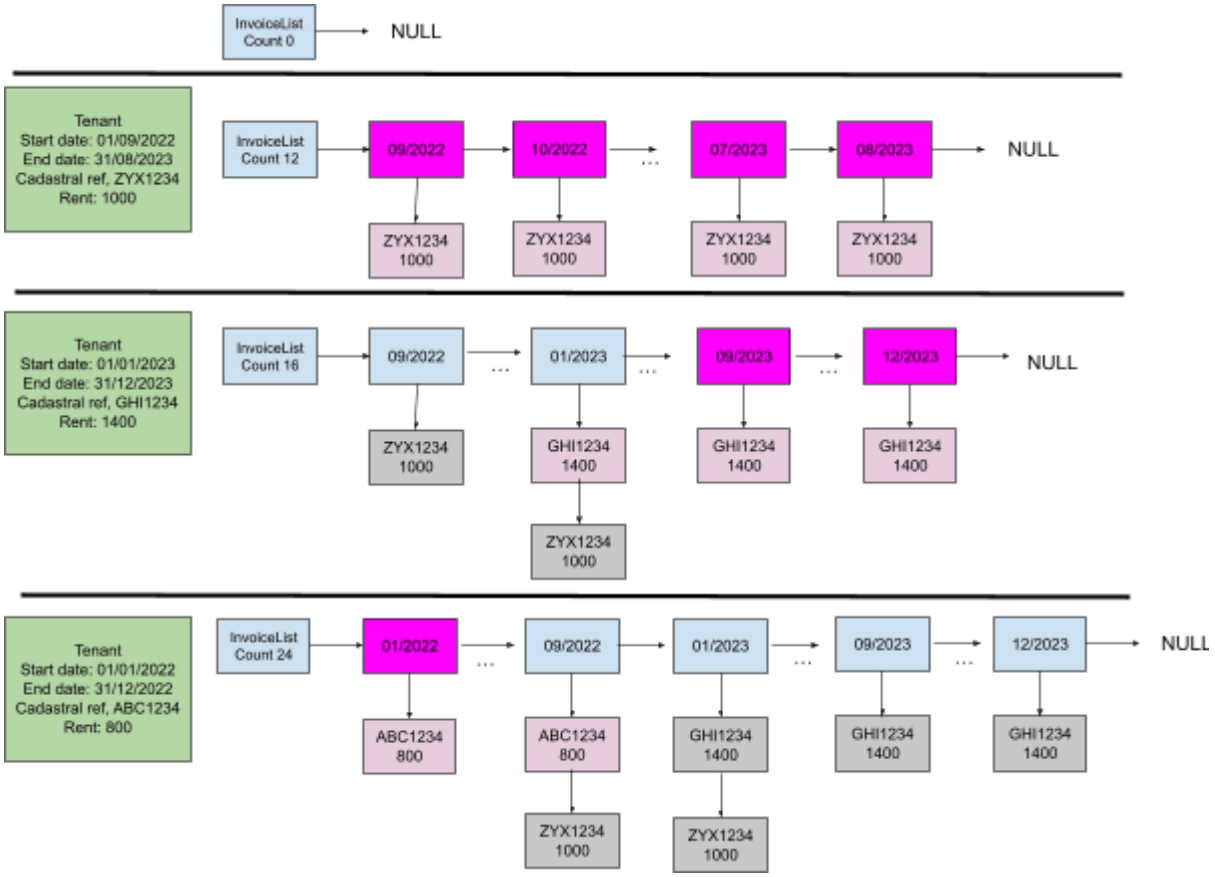
d) El método ***invoiceList_free*** que dado un objeto de tipo **tRentInvoiceData** elimina todos los datos que contiene.

En el código proporcionado también encontraréis la definición e implementación de otras funciones que os pueden ser de utilidad:

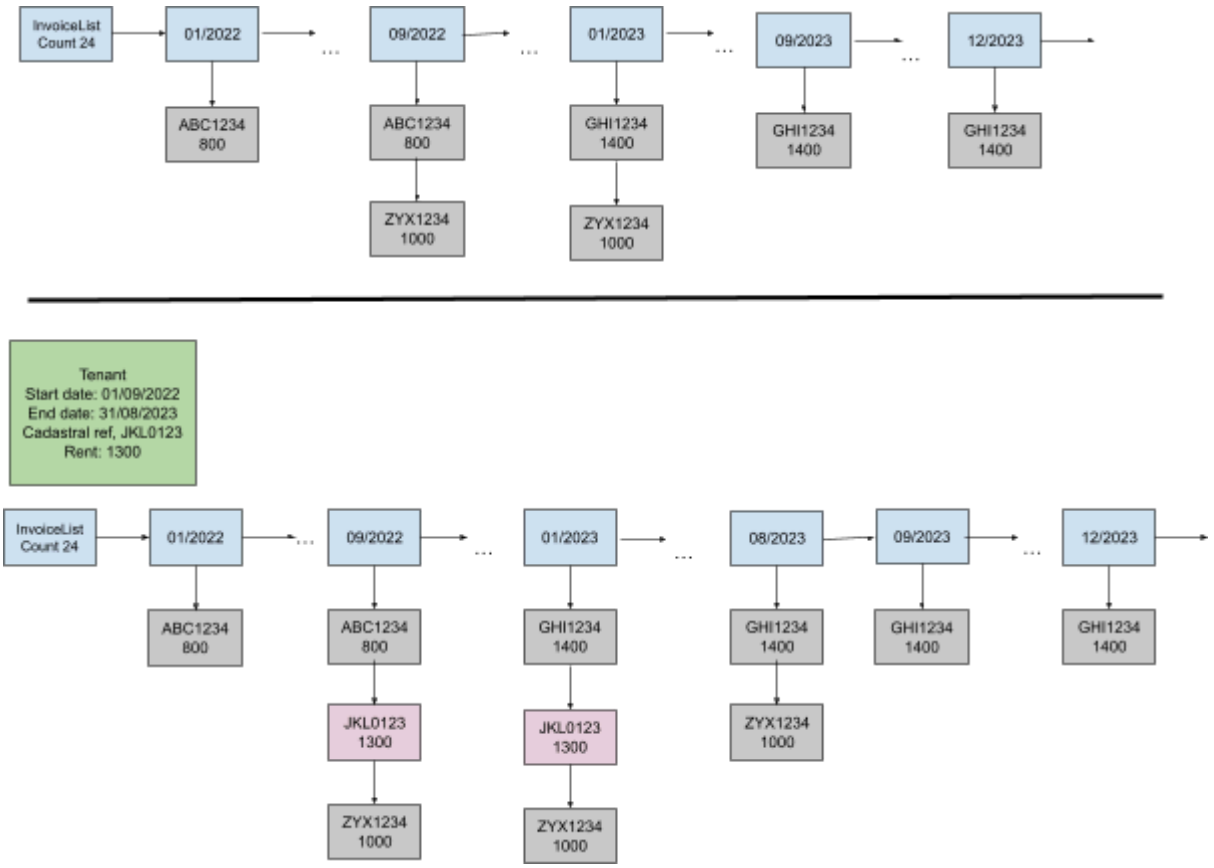
- El método ***invoice_init***, que dado un objeto de tipo **tRentInvoice**, lo inicializa con los datos recibidos como parámetros.
- El método ***invoiceNode_init***, que dado un objeto de tipo **tRentInvoiceNode**, lo inicializa con los datos recibidos como parámetros.
- El método ***monthlyInvoice_init***, que dado un objeto de tipo **tRentInvoiceMonthly,** lo inicializa con el **tDate** recibido como parámetro.
- El método ***monthlyInvoice_free***, que dado un objeto de tipo **tRentInvoiceMonthly**, elimina los datos que contiene.
- El método ***printRentInvoiceData***, que imprime por pantalla la información de un objeto de tipo **tRentInvoiceData**.

- El método ***printRentInvoiceNode***, que imprime por pantalla la información de un objeto de tipo **tRentInvoiceNode**.

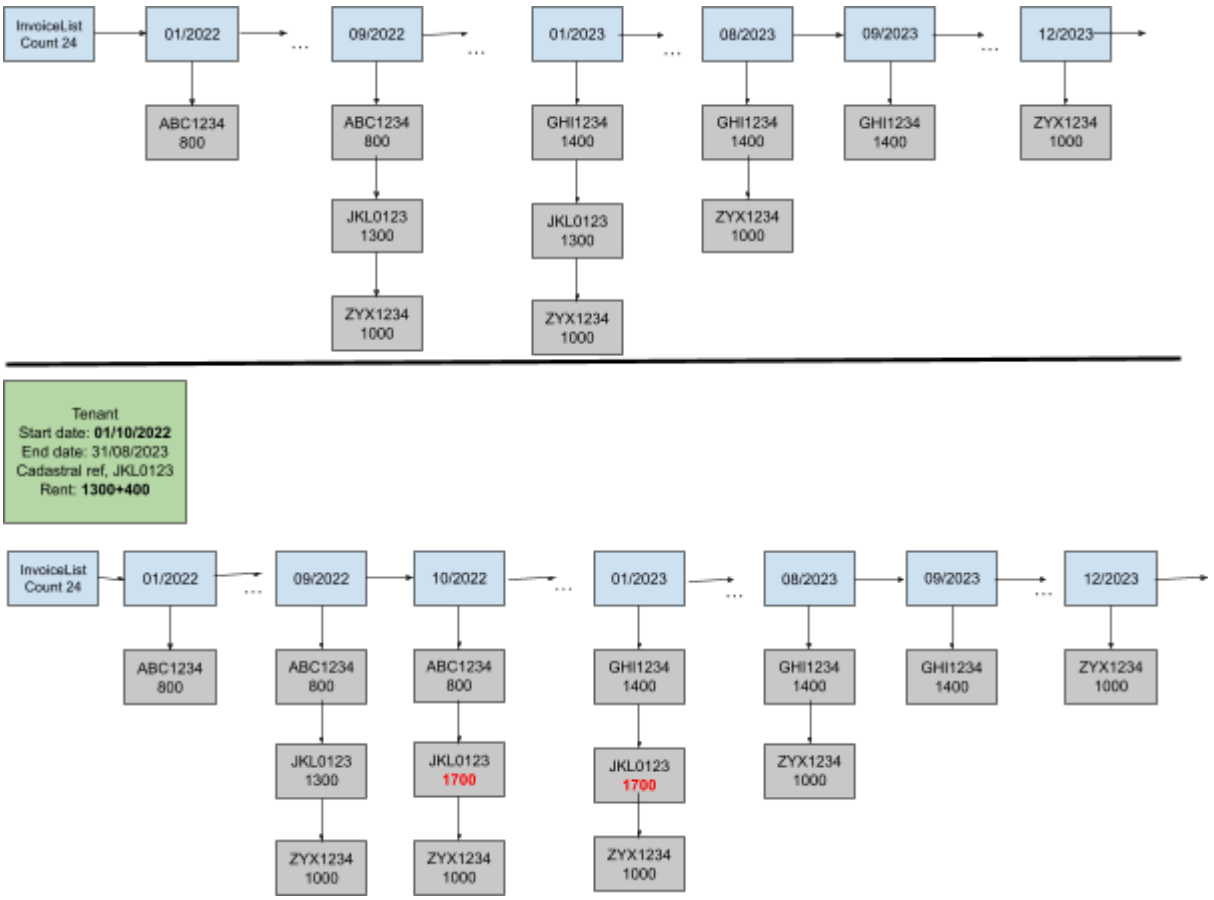
- El método **findMonthlyInvoiceByDate**, que busca una factura mensual dentro de la lista de facturas.



Ejemplo 1: *A partir de una lista vacía, añadimos tres inquilinos.*



Ejemplo 2: *A partir de la lista anterior , añadimos otro inquilino.*



Ejemplo 3: *A partir de la lista anterior , actualizamos los datos de un inquilino.*

Ejercicio 2: Gestión de los oficinas de impuestos [30%]

En el código proporcionado encontraréis los archivos **tax_office.h** y **tax_office.c** con la definición de los tipos de datos para guardar las delegaciones de la agencia tributaria y la implementación de los siguientes métodos:

a) El método ***taxOffice_init*** que dado un objeto de tipo **tTaxOffice**, lo inicializa con el código de la oficina. Inicialmente la oficina tiene una lista de facturas vacía.

b) El método ***taxOffice_free*** que dado un objeto de tipo **tTaxOffice**, elimina los datos que contiene. Para liberar correctamente la memoria se debe tener en cuenta que el método ***invoiceList_free*** del archivo **invoice.c**, si fuera necesario, tendrá que eliminar las oficinas de impuestos que tenga asociados.

c) El método ***taxOfficeList_init*** que dada una lista de oficinas **tTaxOfficeList** la inicializa.

d) El método **check_office_code** que valida que si el código de una oficina es válido, este debe empezar por uno o dos carácteres en mayúsculas que identifican a la provincia y un código numérico de 5 dígitos.
e) El método ***taxOfficeList_insert*** que dada una lista de oficinas **tTaxOfficeList** y el código de la oficina válido de una nueva, añade esta nueva oficina a la lista de oficinas de impuestos. Las oficinas se guardan ordenadas por código de menor a mayor. En caso de que la oficina ya exista, no se hace nada.
f) El método ***taxOfficeList_find*** que dada una lista de oficinas **tTaxOfficeList** y un código de la oficina, devuelve el puntero al centro **tTaxOffice** con ese código. En caso de que no exista ninguna oficina con este código, el método devuelve un valor NULL.
g) El método ***taxOfficeList_free*** que dada una lista de oficinas **tTaxOfficeList** elimina todos los datos que contiene.

Ejercicio 3: Integración en la API [20%]

Finalmente queremos incorporar la gestión de las delegaciones de la agencia tributaria y sus respectivas facturas en la estructura del API, definida e implementada en los ficheros ***api.h*** y ***api.c***. Se pide:

a) Completa la definición del tipo de datos ***tApiData*** del archivo **api.h** para que guarde la lista oficinas de impuestos ***tTaxOfficeList*** definida en el ejercicio anterior.

b) Modifica la función **api_initData** del archivo **api.c** para que inicialice la lista de oficinas de impuestos.

c) Modifica el método **api_addTenant** del archivo **api.c** para que cuando se añada un nuevo tenant:
  i) Si la oficina a la que iría el código postal del contrato de alquiler no existe, añadir esa oficina a la lista de oficinas de impuestos y generar las facturas para las fechas del contrato para la referencia catastral y precio.
  ii) Si ya existe la oficina, actualizar las facturas de la oficina para el nuevo precio desde la fecha de inicio indicada.

d) Modifica el método **api_freeData** del archivo **api.c**, para que elimine todos los datos referentes a las oficinas de impuestos.

e) Implementa el método **api_taxOfficeCount** del archivo **api.c**, que dada una estructura de tipo tApiData, nos devuelva el número de oficinas que contiene.