Práctica 2

Formato y fecha de entrega

Hay que entregar la Práctica antes del día **DD de MMMM a las HH:MMh**. Para la entrega deberéis enviar un fichero en formato **ZIP** de nombre ***logincampus*_pr2** en minúsculas (donde *logincampus* es el nombre de usuario con el que hacéis login en el Campus). El ZIP tiene que contener:

- Workspace CodeLite entero, con todos los ficheros que se piden.

Para reducir la medida de los ficheros y evitar problemas de envío que se pueden dar al incluir ejecutables, hay que eliminar el que genera el compilador. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente (las subcarpetas Menú y Test son las que contienen todos los binarios que genera el compilador).

Hay que hacer la entrega en el apartado de entregas de AC del aula de teoría.

Los estudiantes que participáis en la prueba piloto PROxA y trabajáis con el entorno Vocareum (Aula Isabel Domènech) lo tenéis que entregar directamente en el laboratorio virtual Vocareum haciendo click en el botón submit de vuestro workspace de la PA. Se podrá hacer submit tantas veces como se quiera. Sólo se corregirá la última versión entregada.

Presentación

Esta práctica culmina el proyecto empezado a la práctica 1. Hemos construido una aplicación para gestionar los hoteles y los clientes de una compañía de reservas hoteleras. En esta práctica introduciremos la gestión de las entradas y salidas que hacen el clientes en los diferentes hoteles. Trabajaremos con un TAD lista por manejar las reservas en sus diferentes estados y un TAD cola para representar las colas de checkin y checkout de los clientes en los hoteles.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Conocimientos básicos sobre el uso y la programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Analizar un enunciado y extraer los requerimientos tanto de tipos de datos como funcionales (algoritmos)
- Analizar, entender y modificar adecuadamente código existente
- Saber utilizar tipo de datos abstractos
- Saber utilizar punteros

Recursos

Para realizar esta actividad tenéis a vuestra disposición los siguientes recursos:

- Materiales en formato web de la asignatura
- Laboratorio de C

Criterios de valoración

Cada ejercicio tiene asociada la puntuación porcentual sobre el total de la actividad. Se valorará tanto la corrección de las respuestas como su completitud.

- Los ejercicios en lenguaje C, tienen que compilar para ser evaluados. En tal caso, se valorará:
  - Que funcionen correctamente
  - Que se respeten los criterios de estilo (Véase la Guía de estilo de programación en C que tenéis a la Wiki)
  - Que el código esté comentado (preferiblemente en inglés)
  - Que las estructuras utilizadas sean las adecuadas

Descripción del proyecto

En esta práctica gestionaremos los movimientos de entrada y salida de clientes nos los hoteles de la compañía. Los hoteles, diariamente, tendrán una cola de entrada y salida de clientes que habrá que ir gestionando, utilizando para tal fin, las listas de reservas pendientes, en curso y completadas.

Una reserva de un hotel contendrá la siguiente información:

- Identificador del cliente que hace la reserva
- Número de personas alojadas en esta reserva
- Día de entrada
- Día de salida
- Relación de habitaciones asignadas
- Régimen (alojamiento, alojamiento y desayuno, media pensión o pensión completa)
- Importe total de la estancia.

Por cada hotel, guardaremos la relación de sus reservas mediante tres listas:

- pendingBookings. Es la lista de reservas que tienen como fecha de entrada una fecha futura en el tiempo. Es decir, pendientes de ser ejecutadas.
- currentBookings: Es la lista de reservas que tienen una fecha de entrada pasada en el tiempo pero una fecha de salida futura. Por lo tanto, son reservas en ejecución.
- completedBookings: Es la lista de reservas que tienen una fecha de salida pasada en el tiempo. Es decir, son reservas ya ejecutadas.

Los hoteles organizan sus procesos de checkin y checkout en dos horas diferenciadas. A las 11h se hacen los checkout's y a las 13h se hacen los checkin's. Este comportamiento se modelizará mediante dos colas que representarán las personas que están pendientes de estos procesos de entrar y salida en el hotel:

- checkinQueue. Es la cola de personas que están pendientes del checkin.
- checkoutQueue. Es la cola de personas que están pendientes del checkout.

Siempre se procesarán primero las salidas y después las entradas, de forma que no se empezarán a tratar los elementos de la cola de checkins hasta que la cola de checkouts esté vacía.

- Un checkin consistirá en comprobar que la persona tenía una reserva pendiente de entrada en la fecha actual y, en caso de que sea así, asignarle las habitaciones y mover la reserva de la lista de reservas futuras a la de reservas en curso.
- Un checkout consistirá a comprobar que la persona tenía una reserva en curso que finaliza en la fecha actual y, en caso de que sea así, liberar la habitación (o habitaciones) y mover la reserva de la lista de reservas actuales a la de reservas completadas. Finalmente, se calculará el importe de la factura a pagar por el cliente.

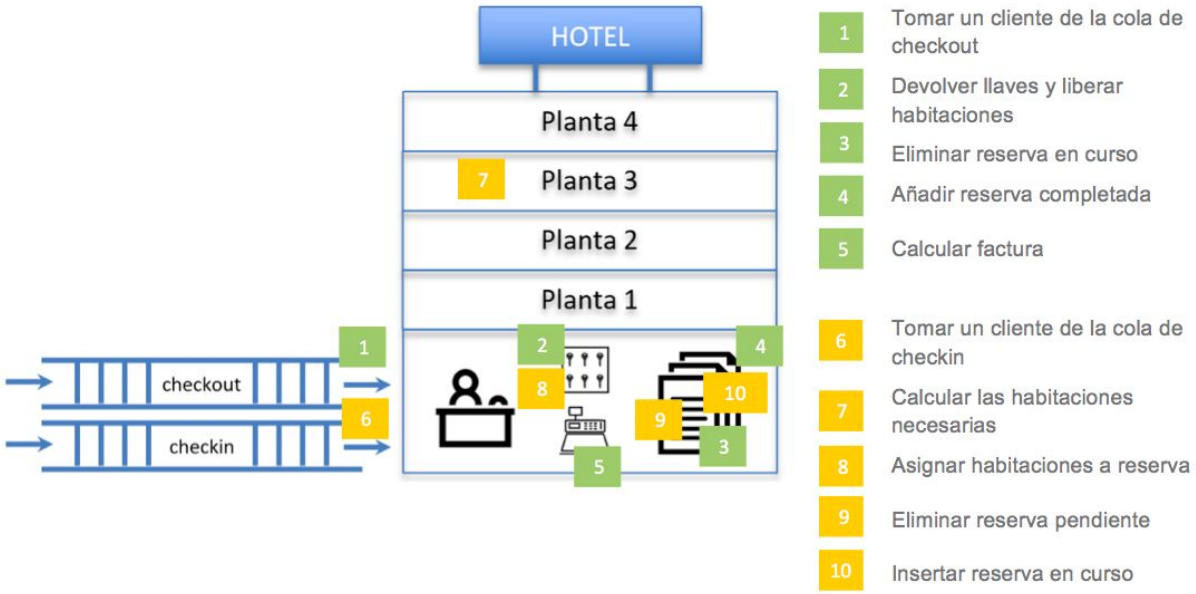
Consideraremos, en esta segunda práctica, que los hoteles gestionados por esta compañía tienen 40 habitaciones que siguen la misma distribución de habitaciones:

- En la primera planta hay 10 habitaciones de uso individual (números de habitación comprendidos entre 101 y 110).
- En la segunda planta hay 10 habitaciones de uso doble (números de habitación comprendidos entre 201 y 210).
- En la tercera planta hay 10 habitaciones de uso triple (números de habitación comprendidos entre 301 y 310).
- En la cuarta planta hay 10 habitaciones de uso cuádruple (números de habitación comprendidos entre 401 y 410).

El hecho que haya el mismo número de habitaciones por planta a pesar de que las superiores son de mayor superficie es debido a que los espacios de uso común de los hoteles se sitúan en las plantas más bajas del edificio (recepción, salones, comedor, gimnasio, etc).

Para simplificar el problema, en el proceso de checkin, se asignarán habitaciones de la medida que convenga para dar cabida a todas las personas incluidas en una misma reserva. Es decir, las habitaciones no están pre-asignadas si no que se asignan el mismo día de la entrada en función de la disponibilidad de habitaciones de las diferentes tipologías.

Esquemáticamente:



En el esquema se pueden apreciar los pasos a seguir en los procesos de checkin (verde) y checkout (amarillo).

Junto con el enunciado se os facilita un nuevo workspace Codelite basado en la solución de la práctica 1, pero con algunos cambios y evoluciones. Este workspace será la base para esta segunda práctica.

Veréis que han aparecido nuevos ficheros: **movements.c y .h** (que recoge las operaciones sobre las reservas), **list.c y .h** (que recoge las operaciones sobre el TAD lista) y, por último, **queue.c y .h** (que recoge las operaciones sobre el TAD cola). Además, se han introducido los nuevos tipos de datos relacionados en data.h :

- **tBooking**, que representa una reserva de un cliente en un hotel en un hotel.
- **tBookingList**, para representar una lista de reservas.
- **tCustomerQueue**, que representa una cola de clientes que esperan para entrar o salir del hotel.
- **tMovement**, que representa los movimientos de un día en un hotel.
- **tMovementTable**, que contiene varios elementos de tipos tMovement.
- El tipo **tHotel** incorpora también una tabla bidimensional que representa la distribución y ocupación de habitaciones dentro del hotel (campos *layout* y occupiedRooms).
- También se incorporan otros tipos auxiliares de apoyo a los anteriores.

Además del código fuente añadido y/o modificado, veréis que aparece también un nuevo fichero de datos: el fichero de movimientos (movements.txt). Este fichero contiene la persistencia de los objetos de tipos *tMovement*.

Tests

El proyecto CodeLite se presenta, como la Práctica 1, con dos modos de ejecución. No obstante, el modo Menú no incorpora ninguna entrada nueva que os permita probar interactivamente la gestión de las reservas. Esto es así porque se pretende que ejecutéis y probéis vuestro programa en modo Test.

En modo Test podréis ir verificando el funcionamiento de vuestro código a medida que completáis los ejercicios. Tened en cuenta que el resultado puede ser correcto aunque el código no esté bien (es decir, el resultado no os da una corrección definitiva) y que el total de tests superados no equivale a la nota de la práctica puesto que el número de tests por apartado no se corresponde con su peso en la calificación.

Enunciado

[50%] Ejercicio 1: Check-in de clientes en un hotel

El procesamiento de los movimientos en un hotel se hace en base a dos procesos diferenciados: la entrada de clientes (checkin) y la salida de clientes (checkout). Esto queda de manifiesto en la acción processMovement (api.c) donde se hacen llamadas a los dos procesos de checkin y checkout. Primero siempre se llama al checkout (porque permite liberar habitaciones que pueden servir a los nuevos clientes) y, después, el checkin.

En este ejercicio nos centraremos en el proceso de checkin. A tal efecto, se pide que completéis estos dos apartados:

a) [15%] Implementad la función ***findCustomerBooking*** (api.c) que, dados una lista de reservas y un identificador de cliente, busca en la lista la primera reserva de la lista hecha por el cliente indicado. En caso de encontrarse la reserva, se devolverá el índice que ocupa la reserva en la lista de reservas. En caso de que no se encuentre, se devolverá un valor NO_BOOKING.

b) [35%] Implementad la acción ***processCheckins*** (api.c) que permita seguir las acciones marcadas en el itinerario amarillo en el esquema anteriormente presentado. Se tienen que ir desencolando los clientes de la cola e ir procesando individualmente su checkin. Por cada cliente, se buscará su reserva en la lista de reservas pendientes (pendingBookings) haciendo uso de la función del apartado anterior. Una vez localizada la reserva, se procesará sólo en caso de que la fecha de entrada de la reserva coincida con la actual (que se recibe por parámetro en la acción). El procesamiento de la reserva consistirá en eliminarla de la lista de reservas en pendientes e insertarla al principio de la lista de reservas en curso. Hará falta, además calcular y asignar qué (y cuántas habitaciones) serán necesarias para alojar las personas incluidas en la reserva.

NOTAS:

- Para saber si un cliente existe en la tabla de clientes podéis buscarlo a partir de su identificador con la ayuda de la función *customerTable_find*.
- Tenéis disponibles operaciones para trabajar con listas de reservas en list.c.
- Podéis calcular las habitaciones que hacen falta para dar cobertura a la reserva haciendo una invocación a la acción *assignRoomsForBooking* (api.c).

[50%] Ejercicio 2: Check-out de clientes en un hotel

Continuando el análisis del procesamiento de los movimientos de un hotel, nos centramos ahora en el proceso de checkout. A tal efecto, se pide que resolváis los siguientes apartados:

a) [15%] Implementad la función ***calculatePrice*** (api.c) que, a partir de un hotel, una reserva y una fecha actual, calcule el importe a pagar por la reserva. Para calcular el importe hay que tener presente que se paga una cantidad por cada habitación reservada y otra por cada persona alojada dentro de estas habitaciones. En cuanto a la habitación, el hotel marca un precio de referencia que corresponde al de la habitación doble. A partir de este precio, la individual se calcula como un 0.75 del precio de la doble. La triple tiene un factor de 1.5 veces el precio de la doble, mientras que en el caso de la cuádruple, el factor es de 1.75. Después, por cada persona, se paga según el régimen: 0€ adicionales si el régimen es de sólo dormir, 10€ si es alojamiento y desayuno, 25€ si es media pensión y 40€ si es pensión completa. En caso de salidas posteriores a la fecha prevista, se carga un 20% adicional al total calculado (independientemente del tiempo excedido).

b) [35%] Implementad la acción ***processCheckouts*** (api.c) que permita seguir las acciones seguidas con itinerario verde según el esquema anteriormente presentado. Se tienen que ir desencolando los clientes y tratándolos en la orden en que se encuentran en la cola. Por cada cliente, se buscará su reserva en la lista de reservas en curso (currentBookings) haciendo uso de la función del apartado a) del ejercicio 1. Una vez localizada la reserva, se procesará sólo en caso de que la fecha de salida de la reserva coincida con el actual (que se recibe por parámetro en la acción). El procesamiento de la reserva consistirá en eliminarla de la lista de reservas en curso e insertarla al principio de la lista de reservas completadas. Hará falta, además, calcular el precio de la estancia y liberar las habitaciones ocupadas (campos price y assignedRooms de tBooking).

NOTAS:

- Para saber si un cliente existe en la tabla de clientes podéis buscarlo a partir de su identificador con la ayuda de la función *customerTable_find*.
- Tenéis disponibles operaciones para trabajar con listas de reservas en list.c.
- Podéis calcular el precio de la estancia con la ayuda de la función *calculatePrice* y la liberación de habitaciones la podéis hacer invocando *freeRoomsOfBooking*.
- Para calcular el número de días que hay entre dos fechas (necesario para el primer apartado), podéis realizar el cálculo que hace la función que os presentamos en la página siguiente. Observad que, para poder hacer estos cálculos, deberéis realizar un include <time.h> al principio del fichero.

```
int numberOfDays(tDate d1, tDate d2)
{
    double seconds;
    struct tm t1= { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    struct tm t2= { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    t1.tm_mday= d1.day;
    t1.tm_mon= d1.month-1;
    t1.tm_year= d1.year-1900;
    t2.tm_mday= d2.day;
    t2.tm_mon= d2.month-1;
    t2.tm_year= d2.year-1900;
    seconds= fabs( difftime( mktime(&t1), mktime(&t2) ) );
    return (int)(seconds / 86400.0);
}
```