Programación de Sistemas y Concurrencia

Control Bloque 1, Temas 1 - 2
Curso 2020-2021

Informática A

APELLIDOS_____ NOMBRE_____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN_____

Descripción del sistema

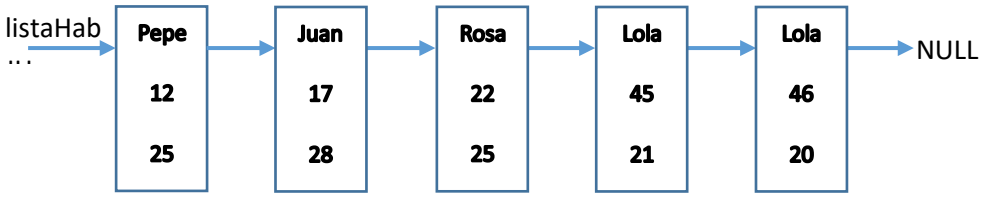
Se quiere almacenar en una estructura dinámica los clientes que están alojados en un hotel. Para ello, el ejercicio se divide en dos partes. En la primera parte, se almacenan los clientes como una lista enlazada ordenada por el número de habitación que ocupa cada cliente. En la segunda parte, los clientes se organizan también según la planta del hotel en la que está su habitación.

Primera Parte

En esta primera parte se debe implementar una lista enlazada con los tipos y prototipos que se encuentran en el fichero "**Planta.h**" y que se describen a continuación:

```
typedef struct Nodo *ListaHab;
struct Nodo {
	char nombre[20];
	unsigned numHab;
	unsigned fechaSalida;
	ListaHab sig;
};
```

Una lista de habitaciones es una estructura enlazada en la que cada nodo tiene el nombre del cliente, el número de la habitación que ocupa y la fecha de salida del cliente. Los nodos se ordenan según el valor creciente del número de habitación tal y como aparece en la siguiente figura.



En el fichero **Planta.c** deben implementarse las siguientes funciones. Recordad que para manejar los string hay que utilizar las funciones de la librería "**string.h**".

```
/**
 * crea una lista de habitaciones vacia
 */
(0.5 pto) void crear(ListaHab *lh);

/**
 * En esta función se añade a la lista, la habitación con número nh,
 * cliente "nombre" y fecha de salida "fs".
 * Si ya existe una habitación numerada con nh en la lista, se actualizan sus campos
 * con los nuevos valores ("nombre" y "fs").
 * Si no existe ninguna habitación con ese número, se inserta un nuevo nodo de manera que
 * la lista siempre esté ordenada con respecto a los números de las habitaciones (de
 * menor a mayor)
 */
(1.5 pto) void nuevoCliente(ListaHab *lh,unsigned nh,char *nombre,unsigned fs);
```

```
/**
 * Escribe por la pantalla la información de cada una de las habitaciones
 * almacenadas en la lista.
 * El formato de salida debe ser:
 * \t (tabulador) habitacion "nh" ocupada por "nombre" con fecha de salida "fs"
 */
(0.5) pto-void imprimir(ListaHab lh);

/**
 * Borra todos los nodos de la lista y la deja vacía
 */
(1 pto) void borrar(ListaHab *lh);

/**
 * Borra todos las habitaciones cuya fecha de salida es fs.
 */

(1.5) pto-void borrarFechaSalida(ListaHab *lh,unsigned fs);
```

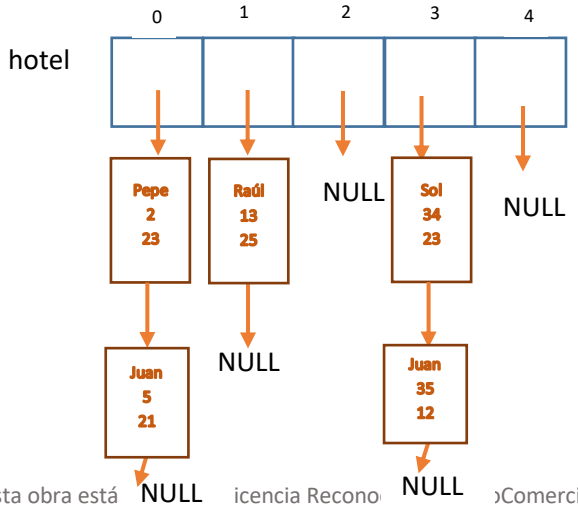
En el fichero **driverPlanta.c**, podéis encontrar un fichero main para probar las funciones anteriores.

La salida de la ejecución de este fichero es:

```
        habitación 1 ocupada por Mara Jaime con fecha de salida 26
        habitación 2 ocupada por Susana Cintra con fecha de salida 23
        habitación 3 ocupada por Pepa Ruiz con fecha de salida 25
        habitación 4 ocupada por Carmen Pasa con fecha de salida 26
        habitación 5 ocupada por Juan Lena con fecha de salida 22
        habitación 7 ocupada por Felipe Moreno con fecha de salida 26


Ahora borramos los que salen el día 26

        habitación 2 ocupada por Susana Cintra con fecha de salida 23
        habitación 3 ocupada por Pepa Ruiz con fecha de salida 25
        habitación 5 ocupada por Juan Lena con fecha de salida 22


Ahora borramos toda la lista
```

Segunda Parte

En la segunda parte del ejercicio, utilizamos los tipos y funciones de la primera parte para almacenar los clientes del hotel organizados según la planta en la que se encuentra su habitación. De esta forma, un hotel es un array de listas de habitaciones con tantas componentes como plantas tiene el hotel, tal y como aparece en la figura. Así, en la componente 0, aparecen las clientes que están en una habitación desde 0 hasta 9, en la componente 1, los que están en una habitación numerada con 10 hasta 19, y así sucesivamente.



En el fichero **Hotel.h** se encuentran los prototipos de las funciones que hay que implementar en el fichero **Hotel.c**. Tened en cuenta que, para implementar las funciones de **Hotel.h** basta con llamar adecuadamente a las funciones de **Planta.h**.

```
/**
 * crea un hotel como un array de NPlantas cada una de ellas una lista de habitaciones
 * vacía
 */
(0.75 pto) void crearHotel(ListaHab* h,unsigned NPlantas);
/**
 * añade a la lista de clientes del hotel h que tiene NPlantas un nuevo cliente en
 * la habitacion nh (en la planta nh/10) con el nombre y fecha de salida de los parámetros
 * El cliente no se añade si el nh/10 no es una planta del hotel o si la fecha no es un valor
 * de un día del mes (entre 1 y 31)
 */
(0.75 pto) void nuevoClienteHotel(ListaHab *h,unsigned NPlantas,unsigned nh,char *nombre,unsigned
fs);
/*
 * Se imprimen todos los clientes del hotel h que tiene NPlantas
 */
(0.75 pto) void imprimirHotel(ListaHab *h,unsigned NPlantas);
/**
 * Se borran todos los clientes del hotel h que tiene NPlantas
 */
(0.75 pto) void borrarHotel(ListaHab *h,unsigned NPlantas);
```

En el fichero, **driverHotel.c** podéis encontrar una función main para probar estas funciones. En ese fichero también se encuentra el prototipo de la función **`cargarHotel`** que lee la información de los clientes de un hotel de un ***fichero de texto*** y la almacena en el array hotel. Entre los ficheros que se suministran para el examen se encuentra el fichero de texto DatosHotel.txt, para que podáis probar la implementación

```
/**
 * Se almacena en el hotel con nPlantas, los clientes almacenados en el fichero de texto nombre.
 * En el fichero los clientes vienen con el siguiente formato
 * Nombre1 numHab1 fechaSal1
 * Nombre2 numHab2 fechaSal2
 * ...
 * Observa que en el fichero viene antes el nombre del cliente que el número
 * de habitación (al contrario que en la funcion nuevoClienteHotel)
 */
(2 pto) void cargarHotel(char* nombre, ListaHab* hotel, unsigned nPlantas)
```

La salida por pantalla de la ejecución de este programa es:

```
Planta numero 0
        habitación 2 ocupada por Yolanda Cabo con fecha de salida 23
        habitación 4 ocupada por Carmen Valero con fecha de salida 22
Planta numero 1
        habitación 13 ocupada por Paco Timo con fecha de salida 25
Planta numero 2
        habitación 22 ocupada por Lorena Campos con fecha de salida 26
        habitación 24 ocupada por Rosa Moral con fecha de salida 27
Planta numero 3
        habitación 32 ocupada por Marcos Navarro con fecha de salida 26
Planta numero 4
        habitación 44 ocupada por Fernando Vera con fecha de salida 27
        habitación 45 ocupada por Abel Justo con fecha de salida 27


Borramos los que salen el 27
```

```
Planta numero 0
         habitación 2 ocupada por Yolanda Cabo con fecha de salida 23
         habitación 4 ocupada por Carmen Valero con fecha de salida 22
Planta numero 1
         habitación 13 ocupada por Paco Timo con fecha de salida 25
Planta numero 2
         habitación 22 ocupada por Lorena Campos con fecha de salida 26
Planta numero 3
         habitación 32 ocupada por Marcos Navarro con fecha de salida 26
Planta numero 4


Ahora borramos el hotel

Planta numero 0
Planta numero 1
Planta numero 2
Planta numero 3
Planta numero 4


Hotel borrado



Ahora cargamos los datos del fichero DatosHotel.txt

Planta numero 0
         habitación 1 ocupada por Sonia-Palermo con fecha de salida 22
         habitación 4 ocupada por Valeria-Romano con fecha de salida 25
Planta numero 1
         habitación 11 ocupada por Pedro-Parco con fecha de salida 24
Planta numero 2
         habitación 21 ocupada por Rosa-Vera con fecha de salida 25
         habitación 22 ocupada por Carmen-Campos con fecha de salida 3
Planta numero 3
         habitación 33 ocupada por Juan-Melero con fecha de salida 21
Planta numero 4
         habitación 43 ocupada por Carmelo-Pastrana con fecha de salida 22
```

ANEXO

Los prototipos de las funciones para manipular strings (incluidas en <string.h>) son:

char* strcpy(char *s1, char *s2): Copia los caracteres de la cadena s2 (hasta el carácter '\0', incluido) en la cadena s1. El valor devuelto es la cadena s1.

Los prototipos de las funciones para **manipulación de ficheros de texto** (incluidos en <stdio.h>) son los siguientes (se dan por conocidos los prototipos de las funciones de <stdlib.h> que necesites, como free o malloc):

FILE *fopen(const char *path, const char *mode): Abre el fichero especificado en el modo indicado ("rb"/ y "wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

int fclose(FILE *fp): Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

int fscanf(FILE *stream, const char *format, ...): Lee del fichero stream los datos con el formato especificado en el parámetro format, el resto de parámetros son las variables en las que se almacenan los datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído con éxito.