UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

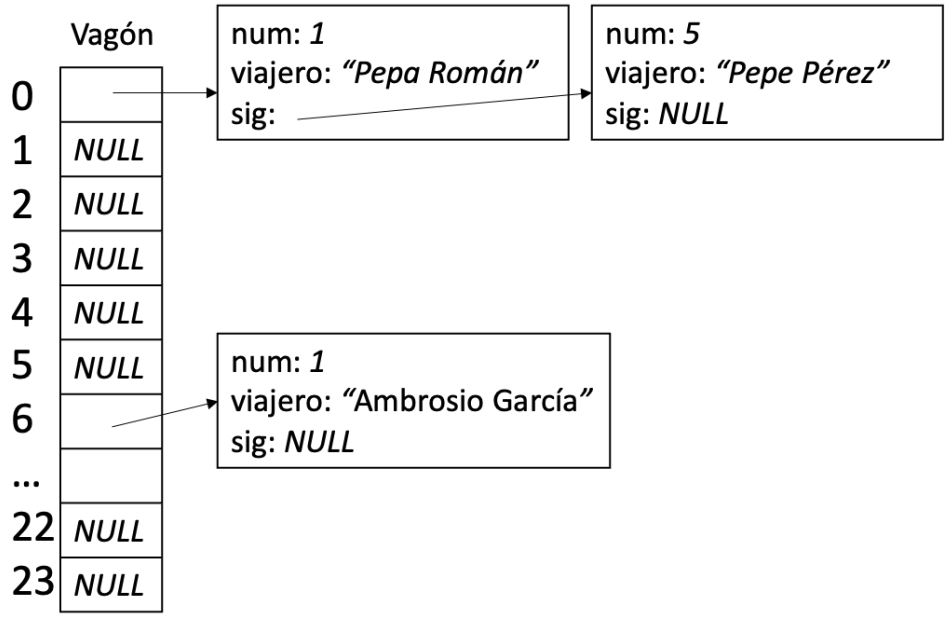
Programación de sistemas y concurrencia

ANDALUCÍA TECH
Campus de Excelencia Internacional
Campus of International Excellence

Computadores A, Informática D, Software D, Matemáticas + Informática D

En este ejercicio, se va a implementar un sistema de control de asientos de un tren. Por temas de seguridad en caso de accidente, se debe tener registrado dónde está sentado cada viajero. En todo momento, un viajero puede solicitar al revisor el intercambio de asiento con otro viajero. Un tren está compuesto como máximo por 24 vagones.

La siguiente figura describe el sistema a implementar:



El sistema se va a componer de un array de 24 vagones. Utilizaremos las siguientes definiciones de tipos para modelar un asiento y un vagón (se incluyen en el fichero Vagon.h):

```
typedef struct Asiento * Vagon;
struct Asiento{
        unsigned num;
        char viajero[30];
        Vagon sig;
};
```

De este modo, un tren se define como es un array de vagones (como podéis ver en el fichero Principal.c). Dentro de un vagón, los asientos se almacenarán **por orden creciente del número del asiento.** El vagón solo almacenara los asientos que están ocupados por un viajero.
Para desarrollar este ejercicio, se proporciona un archivo de cabecera *Tren.h* completo, uno para la implementación incompleto *Tren.c*, y un archivo de código fuente *Principal.c* con *asserts* para realizar algunas pruebas básicas de funcionamiento y que contiene el array de vagones ya definido.

Se deben implementar las siguientes funciones en el archivo *Tren.c*:

```
/// @brief Inicializa el tren creado en el Principal que tiene maximoVagones (Vagon
*tren) con todos los vagones vacío.
/// @param tren Array que representa el tren.
/// @param maximoVagones Número de vagones que tiene el tren (tamaño del array).
/// 0.25 pts
void inicializarTren(Vagon * tren, int maximoVagones);
/// @brief Inserta los datos de un nuevo pasajero. Si el asiento está libre se lo
asigna y si está ocupado ignora la petición.
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente, en el
que se debe insertar.
/// @param numeroVagon Vagón en el que se quiere sentar el pasajero.
/// @param numeroAsiento Asiento dentro del vagón en el que se quiere sentar el
pasajero.
/// @param nombre Nombre del pasajero.
/// @return Si el asiento ya está ocupado, devuelve -1, si no, devuelve 0.
/// 1.75 pts
int entraPasajero(Vagon * tren, unsigned numeroVagon, unsigned numeroAsiento, char *
nombre);


/// @brief Muestra por pantalla los vagones ocupados mostrando en cada línea los datos
de un pasajero. Por ejemplo:
/*
Vagon 0:
Asiento 2, Carlos García
Asiento 4, Ismael Canario
Vagon 2:
Asiento 1, Macarena Sol
*/
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param maximoVagones Máximo de vagones que tiene el tren.
/// 0.75 pt
void imprimeTren(Vagon * tren, unsigned maximoVagones);


/// @brief El pasajero abandona el tren (es eliminado de la estructura).
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param numeroVagon Vagón en el que se está el pasajero que abandona el tren.
/// @param numeroAsiento Asiento en el que se está el pasajero que abandona el tren.
/// @return Devuelve 0 si el pasajero abandona el tren y -1 si no había pasajero en el
vagón y asiento indicados.
/// 1.5 pts
int salePasajero(Vagon * tren, unsigned numeroVagon, unsigned numeroAsiento);

/// @brief Intercambia dos pasajeros. Para ello es suficiente intercambiar el NOMBRE
del viajero que está en el asiento \p numeroAsiento1 del vagón\p numeroVagon1, con el
del viajero que está en el asiento \p numeroAsiento2 del vagón \p numeroVagon2.
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param numeroVagon1 Vagón en el que se está el pasajero 1 que quiere cambiarse de
sitio.
/// @param numeroAsiento1 Asiento en el que se está el pasajero 1 que quiere cambiarse
de sitio.
```

```
/// @param numeroVagon2 Vagón en el que se está el pasajero 2 que quiere cambiarse de
sitio.
/// @param numeroAsiento2 Asiento en el que se está el pasajero 2 que quiere cambiarse
de sitio.
/// @return Si algún asiento no está ocupado, devuelve -1. Si se puede realizar el
cambio, devuelve 0.
/// 1.75 pts
int intercambianAsientos(Vagon * tren, unsigned numeroVagon1, unsigned
numeroAsiento1,unsigned numeroVagon2, unsigned numeroAsiento2);


/// @brief El tren llega a la última parada y bajan todos los pasajeros del tren. El
tren debe quedar vacío.
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param maximoVagones Maximo de vagones que tiene el tren.
/// 1 pt
void ultimaParada(Vagon * tren, unsigned maximoVagones);

/// @brief Guarda en fichero de TEXTO los datos de los pasajeros en el tren. El formato
del fichero de texto será el siguiente, una primera línea con el siguiente texto:
// Vagon;Asiento;Nombre
// Tras esta línea, incluirá una línea por cada pasajero, ordenados por vagón primero y
luego por número de asiento.
// 0;2;Carmen Garcia
// 0;3;Pepe Perez
// 1;5;Adela Gamez
// 1;7;Josefa Valverde
/// @param filename Nombre del fichero en el que se van a almacenar los datos de los
pasajeros del tren.
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param maximoVagones Máximo de vagones que tiene el tren.
/// 1.5 pts
void almacenarRegistroPasajeros(char *filename, Vagon * tren, unsigned maximoVagones);


/// @brief Algunas estaciones están automatizadas y proporcionan un fichero con los
pasajeros que van a entrar en un vagón en su parada.
// Esta función carga los pasajeros que están en el fichero BINARIO filename en el
// vagón numeroVagon. Se asume que los pasajeros almacenados en el fichero no van a
// sentarse en asientos previamente ocupados.
// El fichero binario almacena la información de cada pasajero con la siguiente
// estructura:
// - Un entero con el número de asiento.
// - La cadena de caracteres con el nombre.
/// @param filename Nombre del fichero que contiene los datos de los pasajeros del
vagon.
/// @param tren Array con los vagones y pasajeros que tiene el tren actualmente.
/// @param numeroVagon Vagon del que se van a importar los pasajeros.
/// 1.5 pts

void importarPasajerosVagon(char *filename, Vagon * tren,unsigned numeroVagon);
```

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.

int strcmp(char *s1, char *s2): Devuelve 0 si las dos cadenas son iguales, <0 si s1 es menor que s2, y >0 si s1 es mayor que s2.

size_t strlen(const char *s): Calcula el número de caracteres de la cadena apuntada por s. *Esta función no cuenta el carácter '\0' que finaliza la cadena*.

Los prototipos de las funciones para **manipulación de ficheros binarios** (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"/"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.

LECTURA/ ESCRITURA TEXTO

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.

int fprintf(FILE *stream, const char *format, …): Escribe en el 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 que hay que escribir. La función devuelve el número de variables que se han escrito con éxito.

LECTURA/ ESCRITURA BINARIA

unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream): Lee *nmemb* datos, cada uno de tamaño *size* bytes, del fichero *stream* y los almacena en la dirección apuntada por *ptr*. Devuelve el número de elementos leídos.

unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream): Escribe *nmemb* datos, cada uno de tamaño *size* bytes, en el fichero *stream*. Los datos se obtenien desde la dirección apuntada por *ptr*. Devuelve el número de elementos escritos.