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 incidencias en un hospital. El sistema va a permitir al personal sanitario registrar incidencias y evaluar su resolución. Las incidencias tendrán una prioridad, de 0 a 9, siendo 0 las más prioritarias y 9 las menos prioritarias.

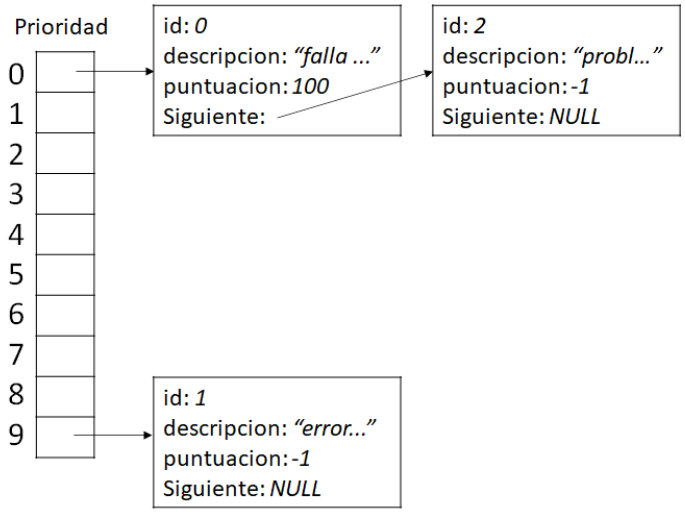
La siguiente estructura describe el sistema a implementar:

```
typedef struct Incidencia *ListaIncidencias;
typedef struct Incidencia
{
    int id;                    // Id único para cada incidencia.
    char descripcion[50];      // Descripción de la incidencia.
    int puntuacion;            // Valor entre 0..100. -1 no evaluado.
    ListaIncidencias siguiente;// Puntero a la siguiente incidencia.
} Incidencia;
```

Además, tendremos un array con tamaño igual a 10, para para tener listas separadas según la prioridad (ya está declarado en el main.c).

```
int t = 10;
ListaIncidencias arrayIncidencias[t];
```

Dentro de la lista, las incidencias se almacenarán **por orden de llegada.** Cada incidencia tendrá un identificador ***id***. La gestión de este campo ***id*** se realizará con la variable ***contId*** (ya está declarada en el Incidencias.c), cada vez que una nueva incidencia se genere, se le asignará contId y se incrementará en uno. La siguiente figura muestra de forma visual la estructura usada:



Para desarrollar este ejercicio, se proporciona un archivo de cabecera *Incidencias.h* completo, uno para la implementación incompleto *Incidencias.c*, y un archivo de código fuente *main.c* con *asserts* para realizar algunas pruebas básicas de funcionamiento. Se deben implementar las siguientes funciones en el archivo *Incidencias.c*:

```
// Inicializa el array de incidencias ya creado en el main (ListaIncidencias *array)
con sus elementos a nulo. El array tiene tamaño t.
// Inicializa la variable contId a cero.
// 0.25 pts.
void inicializarListaIncidencias(ListaIncidencias *array, int t);

// Inserta una nueva incidencia con la prioridad y descripción dada. Asumimos que no
van a insertar una duplicada, y que el array tiene longitud para almacenar la
prioridad dada. Esta función devuelve el id de la incidencia generada. Recuerda que el
array ya está creado en el main.
// 1.75 pts.
int insertarIncidencia(ListaIncidencias *array, int prioridad, char *descripcion);

// Muestra las incidencias por orden de prioridad, primero las más prioritarias. Se
debe mostrar la prioridad, su descripción, y su evaluación (puntuación).
// [Prioridad0 – id0] Descripción1 Sin Evaluar
// [Prioridad0 – id1] Descripción2 Evaluada: 3
// [Prioridad1 – id2] Descripción3 Evaluada: 4
// t es el tamaño del array.
// 1.0 pt.
void mostrarIncidencias(ListaIncidencias *array, int t);

// Libera toda la memoria y deja el array de incidencias vacío. Reinicia contId a 0.
// t es el tamaño del array.
// 1 pt.
void destruirIncidencias(ListaIncidencias *array, int t);

// Cambiar prioridad a una incidencia existente. Se recomienda hacer una función
auxiliar que devuelva la prioridad de una incidencia dado su id.
// t es el tamaño del array.
// 1.75 pt.
void cambiarPrioridad(ListaIncidencias *array, int id, int nuevaPrioridad, int t);

// Establecer la evaluación (puntuación) con valor valorEvaluacion a la incidencia con
id existente.
// t es el tamaño del array.
// 0.5 pt.
void evaluarIncidencia(ListaIncidencias *array, int id, int valorEvaluacion, int t);

// Guarda en un fichero de texto los datos de las incidencias almacenadas en la lista
de incidencias. El formato del fichero de texto será el siguiente, primero tendrá una
cabecera con una descripción de los campos. Tras esta cabecera, una línea por cada
incidencia, ordenadas por prioridad primero y luego por antigüedad (las más antiguas
primero). En caso de no estar evaluada una incidencia, el campo valor será -1;
```

```
// Prioridad;Descripcion;Puntacion;
// 0;Puerta 002 no cierra correctamente;-1;
// 0;Puerta 004 no cierra correctamente;5;
// 9;Puerta 904 no cierra correctamente;100;


// t es el tamaño del array.
// 1.75 pts.
void guardarRegistroIncidencias(char *filename, ListaIncidencias *array, int t);


// Lee de fichero binario los datos de incidencias y los carga para su uso. El array
// puede no estar vacío, recuerda antes borrar todas las incidencias existentes.
// Cada incidencia es almacenada en el fichero con la siguiente estructura:
//- Un entero con la prioridad de la incidencia.
//- Un entero con el tamaño del campo descripción.
//- La cadena de caracteres con la descripción, incluido el carácter terminador '\0'.
//- Un entero con la puntuación.
// Se asume que las incidencias están guardadas por antigüedad, siendo las primeras
// las más antiguas.
// t es el tamaño del array.
// 2.0 pts.
void cargarRegistroIncidencias(char *filename, ListaIncidencias *array, int t);
```

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.