Resolución de problema en C

Structs, Punteros, Matrices, Memoria



Objetivo del ejercicio

Vamos a implementar un sistema simple que:

1. **Guarda** libros en un librero
2. Busca libros por **autor**
3. Encuentra el libro con **mayor puntuación**

Además, hay que completar el procesamiento de eventos en *main.c*

Estructura del proyecto

```
base/
├── main.c → Controla el flujo del programa y procesa los eventos
├── instrucciones.md
├── run.sh → Compila el proyecto usando gcc
├── lib/
│   ├── book.c → Implementa la lógica del librero
│   ├── books.h → Declara funciones y estructuras
│   ├── herramientas.c
│   └── herramientas.h
└── test/
```

books.h

```h
#include <stdbool.h>

typedef struct book {
  char* titulo;
  char* autor;
  float puntuacion;
} Book;

Book*** crear_librero(int n, int m);
Book* book_init(char* titulo, char* autor, float puntuacion);
void free_books(Book*** librero, int n, int m);

bool poner_libro(Book* libro, Book*** librero, int n, int m);
Book* encontrar_por_autor(Book*** librero, int n, int m, char* autor);
Book* encontrar_mayor_puntuacion(Book*** librero, int n, int m);
```

Ya implementadas en el código base

Debemos completarlas en book.c

Book* librero;** → matriz de punteros a libros

Book*	Book*	Book*
Book*	NULL	NULL

¿Cómo se crea el librero?

```c
Book*** crear_librero(int n, int m) {
    Book*** librero = calloc(n, sizeof(Book**)); // Librero con N filas
    for (int i = 0; i < n; i++) {
        librero[i] = calloc(m, sizeof(Book*)); // Cada fila con M libros
    }

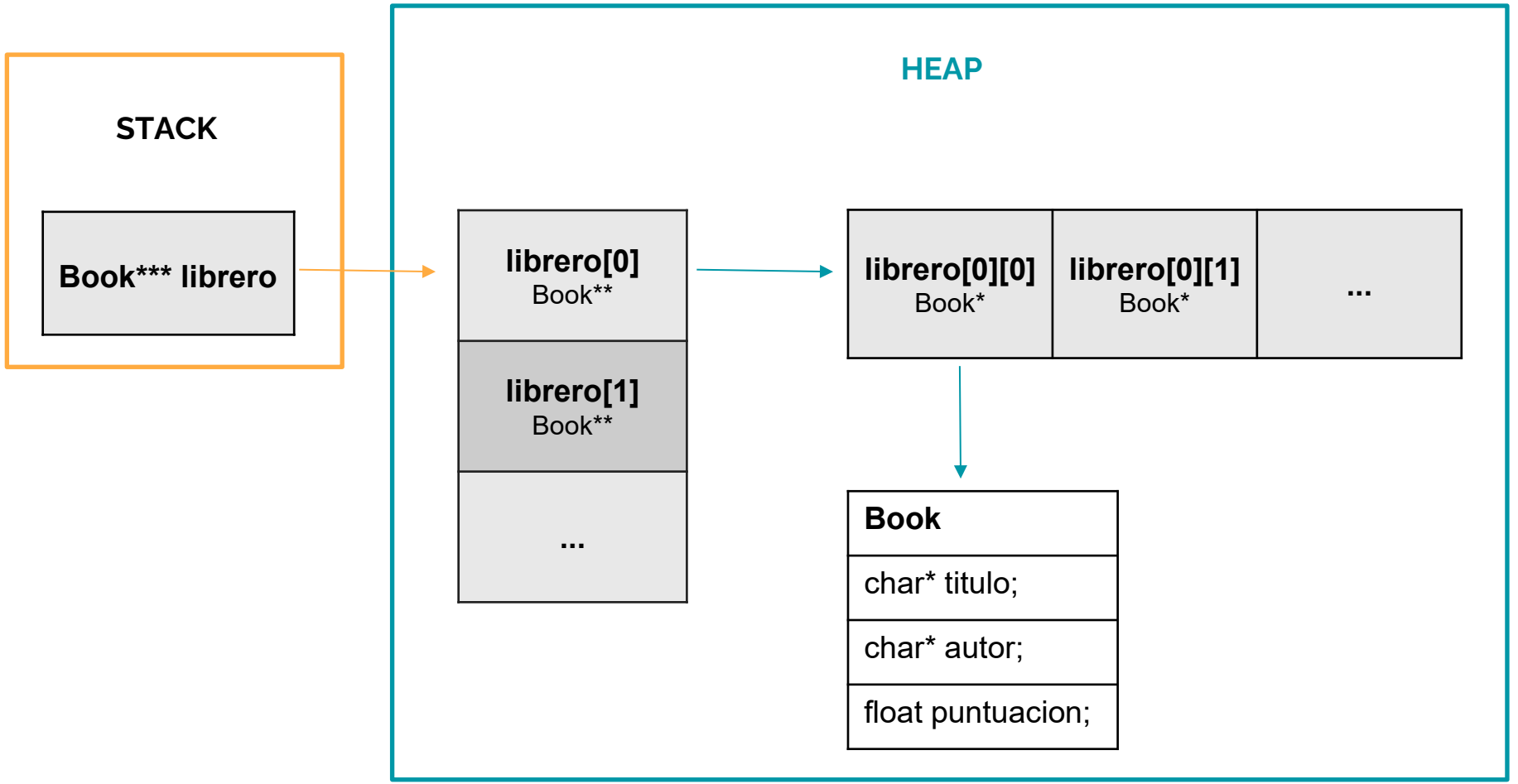
    // Inicializar en NULL los punteros a libros
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            librero[i][j] = NULL;
        }
    }

    return librero;
}
```

book.c

```c
Book*** librero = crear_librero(N, M);
```

main.c


STACK
Book*** librero
HEAP
librero[0]
Book**
librero[1]
Book**
...
librero[0][0]
Book*
librero[0][1]
Book*
...
Book
char* titulo;
char* autor;
float puntuacion;

Eventos

1.
PONER_LIBRO

Objetivo: Agregar un libro al librero en la primera posición disponible.

1. PONER_LIBRO

book.c

```
bool poner_libro(Book* libro, Book*** librero, int n, int m) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      if (librero[i][j] == NULL) { // Asumiendo que un libro vacío tiene titulo NULL
        librero[i][j] = libro; // Colocamos el libro en la posición vacía
        printf("Libro '%s' colocado en la posición [%d][%d]\n", libro->titulo, i, j);
        return true;
      }
    }
  }
  printf("No hay espacio para colocar el libro: %s\n", libro->titulo);
  return false;
}
```

1. PONER_LIBRO

main.c

```c
if (string_equals(command, "PONER_LIBRO")) {
  char titulo[256];
  char autor[256];
  float puntuacion;
  buffer = fscanf(input_file, "%255s %255s %f", autor, titulo, &puntuacion);
  if (buffer != 3) {
      printf("Error reading book information");
      return 1;
  }
  Book* libro = book_init(titulo, autor, puntuacion);
  if (!poner_libro(libro, librero, N, M)) {
    free(libro->titulo);
    free(libro->autor);
    free(libro);
  }
}
```

2.
ENCONTRAR_POR_AUTOR

Objetivo: Buscar en el librero el primer libro escrito por un autor dado.

2. ENCONTRAR_POR_AUTOR

book.c

```c
Book* encontrar_por_autor(Book*** librero, int n, int m, char* autor) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (librero[i][j] != NULL && string_equals(librero[i][j]->autor, autor)) {
                return librero[i][j];
            }
        }
    }
    return NULL; // No se encontró ningún libro del autor
}
```

2. ENCONTRAR_POR_AUTOR

main.c

```
else if (string_equals(command, "ENCONTRAR_POR_AUTOR")) {

  char autor[256];
  buffer = fscanf(input_file, "%255s", autor);

  if (buffer != 1) {
    printf("Error reading author information");
    return 1;
  }

  Book* libro_encontrado = encontrar_por_autor(librero, N, M, autor);

  if (libro_encontrado != NULL) {
      printf("Libro encontrado: %s por %s con puntuación %.1f\n", libro_encontrado-
  } else {
      printf("No se encontró ningún libro del autor: %s\n", autor);
  }
}
```

3. ENCONTRAR_MAYOR_PUNTUACION

Objetivo: Encontrar el libro del librero con la mayor puntuación.

3. ENCONTRAR_MAYOR_PUNTUACION

book.c

```c
Book* encontrar_mayor_puntuacion(Book*** librero, int n, int m) {
    Book* mejor_libro = NULL;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (librero[i][j] != NULL) {
                if (mejor_libro == NULL || librero[i][j]->puntuacion > mejor_libro->puntuacion) {
                    mejor_libro = librero[i][j];
                }
            }
        }
    }
    return mejor_libro; // Retorna el libro con la mayor puntuación o NULL si no hay libros
}
```

3. ENCONTRAR_MAYOR_PUNTUACION

main.c

```
else if (string_equals(command, "ENCONTRAR_MAYOR_PUNTUACION")) {
    Book* mejor_libro = encontrar_mayor_puntuacion(librero, N, M);
    if (mejor_libro != NULL) {
        printf("Libro con mayor puntuación: %s por %s con puntuación %.1f\n",
            mejor_libro->titulo, mejor_libro->autor, mejor_libro->puntuacion);
    } else {
        printf("No hay libros en el librero.\n");
    }
}
```

No olvidar liberar la memoria

En C, toda memoria reservada con malloc/calloc debe liberarse manualmente con free.

book.c

```c
void free_books(Book*** librero, int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (librero[i][j] != NULL) {
                free(librero[i][j]->titulo);
                free(librero[i][j]->autor);
                free(librero[i][j]);
            }
        }
        free(librero[i]);
    }
    free(librero);
}
```

```c
free_books(librero, N, M);
```

main.c

Compilar

- En este ejercicio usamos un script (run.sh) que compila el programa usando **gcc** directamente, en lugar de usar make.
- Esto permite ver explícitamente las etapas de compilación y linking del programa.

run.sh

```
# Compilar
gcc -Wall -Wextra -g -I lib -c main.c -o obj/main.o
gcc -Wall -Wextra -g -c lib/book.c -o obj/book.o
gcc -Wall -Wextra -g -c lib/herramientas.c -o obj/herramientas.o

# Linkear
gcc -o out obj/main.o obj/book.o obj/herramientas.o
```

Cada archivo .c se compila por separado para generar archivos .o

Los archivos .o se combinan para crear el ejecutable final.

Ejecución

```
./run.sh                          # Compila el código
./out test/test_1/INPUT.txt       # Ejecuta con el test 1
```

```
# valgrind
valgrind ./out test/test_1/INPUT.txt
valgrind --leak-check=full ./out test/test_1/INPUT.txt
```

Recuerden que en los talleres compilan con makefile

```
make clean && make
./nombre_ejecutable input.txt output.txt
```



Feedback

