Análisis de los tres códigos C

1. Nombres (Identificadores)

- **Macros y constantes**: `MEMORY_MANAGEMENT_H`, `MEMORY_MANAGEMENT_DISPLAY`.
- **Variables globales/estáticas**:
  - Contadores de memoria: `heap_allocations`, `heap_deallocations`, `stack_allocations`, `stack_deallocations`.
  - `static_var` (segmento de datos) y `bss_var` (BSS).
  - `heap_memory_records` (lista de registros de memoria).
- **Tipos definidos por el usuario**: `genre_t`, `book_t`, `member_t`, `MemoryRecord`.
- **Funciones**: `displayMemoryUsage`, `incrementHeapAllocations`, `incrementHeapDeallocations`, `incrementStackAllocations`, `incrementStackDeallocations`, `addMemoryRecord`, `removeMemoryRecord`, `genreToString`, `addBook`, `findBookById`, `displayBooksRecursive`, `displayBooks`, `addMember`, `issueBook`, `returnBook`, `freeLibrary`, `freeMembers`, `saveLibraryToFile`, `loadLibraryFromFile`, `saveMembersToFile`, `loadMembersFromFile`, `displayMembers`, `searchMember`, `main`.
- **Variables locales**: `choice`, `bookCount`, `memberCount`, `record`, `current`, `to_free`, entre otras.

---

2. Marcos de activación (Activation Records)

- Cada llamada a función crea su marco en la pila con parámetros, variables locales y dirección de retorno.
- Ejemplos: llamadas a `addBook`, `issueBook`, `displayBooksRecursive` (recursión).

---

3. Bloques de alcance

- **Global**: variables fuera de funciones (`heap_allocations`, `heap_memory_records`, `bss_var`).
- **Local**: variables internas a cada función (`new_book`, `genre`, `current`, etc.).
- **Archivo/estático**: `static_var` y funciones internas como `addMemoryRecord`.
- **Preprocesador**: bloques condicionales `#if / #else / #endif` controlan compilación.

---

4. Administración de memoria

- **Heap**: uso de `malloc`, `realloc`, `free` para `book_t`, `member_t`, `issued_books`, `MemoryRecord`.
- **Stack**: variables automáticas en cada función (`main`, `issueBook`).
- **Segmento de datos**: `static_var`, contadores de memoria.
- **BSS**: `bss_var` (global no inicializada).
- Registro y seguimiento de asignaciones/liberaciones con `incrementHeapAllocations`, `incrementHeapDeallocations` y la lista `MemoryRecord`.

---

5. Expresiones

- Asignaciones y cálculos: `bookFound->quantity--`, `memberFound->issued_count++`, `heap_allocations++`.
- Comparaciones: `if ((*current)->pointer == pointer)`, `while (current)`.
- Llamadas a funciones estándar: `printf`, `scanf`, `fopen`, `fgets`, `strcmp`, `malloc`, `free`, `realloc`.

---

6. Comandos (Sentencias)

- Declaraciones y asignaciones de variables.
- Llamadas a funciones de biblioteca estándar y propias.
- Entrada/salida: `printf`, `scanf`, `fprintf`, `fscanf`.
- Manejo de archivos: `fopen`, `fclose`.
- Directivas de preprocesador: `#ifndef`, `#define`, `#endif`, `#if`.

---

7. Control de secuencia

- **Selección**: `if/else`, `switch` (menú principal y `genreToString`).
- **Iteración**: `while`, `for`, `do…while` para recorrer listas enlazadas, menús, lectura de archivos.
- **Recursión**: `displayBooksRecursive` imprime libros llamándose a sí misma.

---

8. Subprogramas

- **Monitoreo de memoria**: `displayMemoryUsage`, `increment*`, `addMemoryRecord`, `removeMemoryRecord`.
- **Gestión de biblioteca**: `addBook`, `findBookById`, `displayBooks`, `addMember`, `issueBook`, `returnBook`, `freeLibrary`, `freeMembers`.
- **Persistencia en archivos**: `saveLibraryToFile`, `loadLibraryFromFile`, `saveMembersToFile`, `loadMembersFromFile`.
- **Interfaz de usuario**: `main` orquesta el flujo del sistema.

---

9. Tipos de datos

- **Primitivos**: `int`, `char`, `size_t`, `void`.
- **Enumeraciones**: `enum genre_t` para géneros de libros.
- **Estructuras**: `struct _book`, `struct _member`, `struct MemoryRecord`.
- **Punteros**: `book_t*`, `member_t*`, `int*`, `void*`, `FILE*`.
- **Arreglos**: `char title[100]`, `char author[100]`, `char name[100]`.

Resumen global

El programa completo (unión de los tres archivos) es un **sistema de gestión de biblioteca** en C que:

- Usa **estructuras enlazadas** para libros y miembros.
- Maneja **memoria dinámica** con seguimiento detallado de asignaciones y liberaciones.
- Implementa **persistencia en archivos** de texto.
- Aplica **control de flujo completo**: selección, iteración y recursión.

- Demuestra interacción entre **segmentos de memoria** (datos, BSS, stack, heap) y programación modular.