```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#define MAX_USERS 50
#define QUEUE_SIZE 5
#define STR_LEN 30


// Structure 1: Queue for Book Reservations
typedef struct {
    int userIds[QUEUE_SIZE];
    int front;
    int rear;
} ReservationQueue;

// Book Node (Linked List)
struct Book {
    int id;
    char title[STR_LEN];
    char author[STR_LEN];
    int isIssued;
    int issuedToUid;
    ReservationQueue reserveQ;
    struct Book *next;
};

// User (Array)
typedef struct {
    int id;
    char name[STR_LEN];
    int hasBook;
    int bookId;
} User;

//GLOBAL VARIABLES
struct Book *head = NULL;   // Head pointer for Book Linked List
User users[MAX_USERS];      // Array to store users
int userCount = 0;          // Current number of users registered


// SECTION 1: QUEUE FUNCTIONS (For Reservation System)
void initQueue(ReservationQueue *q) {
```

```
    q->front = -1;
    q->rear = -1;
}

int isQueueEmpty(ReservationQueue *q) {
    if (q->front == -1)
        return 1; // True
    return 0;     // False
}

void enqueue(ReservationQueue *q, int userId) {
    if (q->rear == QUEUE_SIZE - 1) {
        printf("  [!] Queue Full! Cannot reserve more users.\n");
    } else {
        if (q->front == -1) q->front = 0;
        q->rear++;
        q->userIds[q->rear] = userId;
        printf("  [Success] User %d added to reservation queue.\n", userId);
    }
}


int dequeue(ReservationQueue *q) {
    int item;
    if (isQueueEmpty(q)) return -1;

    item = q->userIds[q->front];
    q->front++;

    // Reset queue if empty
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}


// Find a book pointer by its ID
struct Book* findBook(int id) {
    struct Book *temp = head;
    while (temp != NULL) {
        if (temp->id == id) return temp;
        temp = temp->next;
    }
```

```
    return NULL;
}

// Find user array index by User ID
int findUserIndex(int id) {
    int i;
    for(i = 0; i < userCount; i++) {
        if(users[i].id == id) return i;
    }
    return -1;
}


// LIBRARY FEATURES

void addBook() {
    int id;
    char title[STR_LEN], author[STR_LEN];
    struct Book *newNode;
    struct Book *temp;

    printf("\n--- ADD NEW BOOK ---\n");
    printf("Enter Book ID: ");
    scanf("%d", &id);

    // Validation: Positive ID
    if (id <= 0) {
        printf("  [Error] ID must be a positive number.\n");
        return;
    }

    // Validation: Check Duplicate
    if (findBook(id) != NULL) {
        printf("  [Error] Book ID %d already exists!\n", id);
        return;
    }

    printf("Enter Title: "); scanf("%s", title);
    printf("Enter Author: "); scanf("%s", author);

    // Memory Allocation
    newNode = (struct Book*)malloc(sizeof(struct Book));
    newNode->id = id;
    strcpy(newNode->title, title);
```

```
    strcpy(newNode->author, author);
    newNode->isIssued = 0;
    newNode->issuedToUid = 0;
    newNode->next = NULL;
    initQueue(&newNode->reserveQ); // Initialize the queue for this book

    // Add to Linked List
    if (head == NULL) {
        head = newNode;
    } else {
        temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("  [Success] Book added successfully.\n");
}

void registerUser() {
    int uid;
    char name[STR_LEN];

    printf("\n--- REGISTER USER ---\n");
    if(userCount >= MAX_USERS) {
        printf("  [Error] Maximum user limit reached.\n");
        return;
    }

    printf("Enter User ID: "); scanf("%d", &uid);

    // Validation: Check Duplicate
    if(findUserIndex(uid) != -1) {
        printf("  [Error] User ID %d is already taken.\n", uid);
        return;
    }

    printf("Enter Name: "); scanf("%s", name);

    // Add to Array
    users[userCount].id = uid;
    strcpy(users[userCount].name, name);
    users[userCount].hasBook = 0;
    users[userCount].bookId = 0;
```

```
    userCount++;
    printf("  [Success] User registered successfully.\n");
}

void displayBooks() {
    struct Book *temp = head;

    printf("\n--- ALL BOOKS ---\n");
    printf("ID\tTitle\t\tStatus\t\tWaitlist\n");
    printf("-------------------------------------------------\n");

    while (temp != NULL) {
        printf("%d\t%s\t\t%s\t\t", temp->id, temp->title,
               (temp->isIssued ? "Issued" : "Available"));

        // Show waitlist count
        if(!isQueueEmpty(&temp->reserveQ))
            printf("%d Users", temp->reserveQ.rear - temp->reserveQ.front + 1);
        else
            printf("-");

        printf("\n");
        temp = temp->next;
    }
}

void searchBook() {
    char query[STR_LEN];
    struct Book *temp = head;
    int found = 0;

    printf("\n--- SEARCH BOOK ---\n");
    printf("Enter Title or Author: ");
    scanf("%s", query);

    printf("\nSearch Results:\n");
    while(temp != NULL) {
        if(strcmp(temp->title, query) == 0 || strcmp(temp->author, query) == 0) {
            printf(" -> [%d] %s by %s (%s)\n", temp->id, temp->title, temp->author,
                   (temp->isIssued ? "Issued" : "Available"));
            found = 1;
        }
        temp = temp->next;
    }
```

```
    if(!found) printf("  [Info] No matching books found.\n");
}


// TRANSACTION FUNCTIONS (Issue/Return/Reserve)
void reserveBook() {
    int bid, uid, uIdx, i;
    struct Book *b;

    printf("\n--- RESERVE BOOK ---\n");
    printf("Enter Book ID: "); scanf("%d", &bid);
    printf("Enter User ID: "); scanf("%d", &uid);

    b = findBook(bid);
    uIdx = findUserIndex(uid);

    // Validations
    if(b == NULL) { printf("  [Error] Book not found.\n"); return; }
    if(uIdx == -1) { printf("  [Error] User not found.\n"); return; }

    if(b->issuedToUid == uid) {
        printf("  [Error] You already have this book issued!\n");
        return;
    }

    // Check if user is already in queue
    if (!isQueueEmpty(&b->reserveQ)) {
        for(i = b->reserveQ.front; i <= b->reserveQ.rear; i++) {
            if(b->reserveQ.userIds[i] == uid) {
                printf("  [Error] You are already in the reservation queue.\n");
                return;
            }
        }
    }

    // If book is available and queue is empty, suggest issuing
    if(b->isIssued == 0 && isQueueEmpty(&b->reserveQ)) {
        printf("  [Notice] Book is currently available! Use 'Issue Book' instead.\n");
        return;
    }

    // Add to queue
    enqueue(&b->reserveQ, uid);
}
```

```
void issueBook() {
    int bid, uid, uIdx, nextUser;
    struct Book *b;

    printf("\n--- ISSUE BOOK ---\n");
    printf("Enter Book ID: "); scanf("%d", &bid);
    printf("Enter User ID: "); scanf("%d", &uid);

    b = findBook(bid);
    uIdx = findUserIndex(uid);

    // Basic Validations
    if(b == NULL) { printf("  [Error] Book not found.\n"); return; }
    if(uIdx == -1) { printf("  [Error] User not found.\n"); return; }

    // Check Status
    if(b->isIssued) {
        printf("  [Error] Book is currently issued to User %d.\n", b->issuedToUid);
        return;
    }

    if(users[uIdx].hasBook == 1) {
        printf("  [Error] User already holds a book (ID: %d). Return it first.\n", users[uIdx].bookId);
        return;
    }

    // Check Reservation Queue
    if(!isQueueEmpty(&b->reserveQ)) {
        // Peek at the first person in line
        nextUser = b->reserveQ.userIds[b->reserveQ.front];

        if(nextUser != uid) {
            printf("  [Error] Book is Reserved! Only User %d can issue it now.\n", nextUser);
            return;
        } else {
            // If current user is the one waiting, remove them from queue
            dequeue(&b->reserveQ);
        }
    }

    // Perform Issue
    b->isIssued = 1;
    b->issuedToUid = uid;
```

```
    users[uIdx].hasBook = 1;
    users[uIdx].bookId = bid;
    printf("  [Success] Book Issued to %s.\n", users[uIdx].name);
}

void returnBook() {
    int bid, uIdx, nextUser;
    struct Book *b;

    printf("\n--- RETURN BOOK ---\n");
    printf("Enter Book ID to return: "); scanf("%d", &bid);

    b = findBook(bid);

    if(b == NULL || b->isIssued == 0) {
        printf("  [Error] Invalid Book ID or Book was not issued.\n");
        return;
    }

    // Update User Record
    uIdx = findUserIndex(b->issuedToUid);
    if(uIdx != -1) {
        users[uIdx].hasBook = 0;
        users[uIdx].bookId = 0;
    }

    // Update Book Record
    b->isIssued = 0;
    b->issuedToUid = 0;
    printf("  [Success] Book Returned.\n");

    // Check if anyone is waiting
    if(!isQueueEmpty(&b->reserveQ)) {
        nextUser = b->reserveQ.userIds[b->reserveQ.front];
        printf("  *** ALERT: User %d is waiting for this book! ***\n", nextUser);
    }
}


void main() {
    int choice;

    clrscr();
```

```
do {
        clrscr();
        printf("\n=============================================\n");
        printf("    LIBRARY MANAGEMENT SYSTEM (Validations) \n");
        printf("=============================================\n");
        printf(" 1. Add Book\n");
        printf(" 2. Display All Books\n");
        printf(" 3. Search Book (Title/Author)\n");
        printf(" 4. Register User\n");
        printf(" 5. Issue Book\n");
        printf(" 6. Return Book\n");
        printf(" 7. Reserve Book\n");
        printf(" 0. Exit\n");
        printf("---------------------------------------------\n");
        printf(" Enter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
           case 1:
                addBook();
                break;
           case 2:
                displayBooks();
                break;
           case 3:
                searchBook();
                break;
           case 4:
                registerUser();
                break;
           case 5:
                issueBook();
                break;
           case 6:
                returnBook();
                break;
           case 7:
                reserveBook();
                break;
           case 0:
                printf("Exiting System...\n");
                break;
           default:
                printf("Invalid Choice!\n");
```

```
        }

        if(choice != 0) {
            printf("\nPress any key to continue...");
            getch();
        }

    } while(choice != 0);
}
```