Event Registration System

```
#include <stdio.h> // For input and output functions like printf, scanf
#include <stdlib.h> // For memory management functions like malloc, free
#include <string.h> // For string operations like strcpy, strcmp

// Structure to hold student details (ID, name, and pointer to next student)
struct Student {
    int studentId;
    char name[50];
    struct Student* next;
};

// Structure to represent an event node in a Circular Linked List (CLL)
// Contains event name, head of student list, and pointer to next event
struct Event {
    char eventName[50];
    struct Student* students;
    struct Event* next;
};

// Global pointer to the head of the event CLL, initialized as NULL
struct Event* head = NULL;

/* Step 1: Define Helper Functions for Memory Allocation */

/* Creates a new event node with the given name and initializes its pointers */
struct Event* createEvent(const char* name) {
    struct Event* newEvent = (struct Event*)malloc(sizeof(struct Event));
    if (newEvent == NULL) {
        printf("Memory allocation failed!\n");
```

```
        exit(1);
    }
    strcpy(newEvent->eventName, name);
    newEvent->students = NULL; // No students initially
    newEvent->next = NULL;    // No next event initially
    return newEvent;
}

/* Creates a new student node with the given ID and name */
struct Student* createStudent(int id, const char* name) {
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct Student));
    if (newStudent == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newStudent->studentId = id;
    strcpy(newStudent->name, name); // Fixed: Use correct pointer
    newStudent->next = NULL;      // No next student initially
    return newStudent;
}

/* Step 2: Implement Core Operations on Events */

/* Adds a new event to the CLL, maintaining the circular structure */
void addEvent(const char* eventName) {
    struct Event* newEvent = createEvent(eventName);
    if (head == NULL) {
        head = newEvent;
        head->next = head; // First node points to itself to form a circle
    } else {
        struct Event* temp = head;
```

```
        while (temp->next != head) { // Traverse to the last node
            temp = temp->next;
        }
        temp->next = newEvent;    // Link new event
        newEvent->next = head;    // Complete the circular link
    }
    printf("Event '%s' added successfully.\n", eventName);
}

/* Adds a student to the student list of a specific event */
void addStudentToEvent(const char* eventName, int studentId, const char* studentName) {
    struct Event* temp = head;
    if (temp == NULL) {
        printf("No events available. Add an event first.\n");
        return;
    }
    do {
        if (strcmp(temp->eventName, eventName) == 0) {
            struct Student* newStudent = createStudent(studentId, studentName);
            if (temp->students == NULL) {
                temp->students = newStudent; // First student for this event
            } else {
                struct Student* studentTemp = temp->students;
                while (studentTemp->next != NULL) { // Traverse to the last student
                    studentTemp = studentTemp->next;
                }
                studentTemp->next = newStudent; // Add new student at the end
            }
            printf("Student '%s' (ID: %d) added to event '%s'.\n", studentName, studentId, eventName);
            return;
        }
```

```
        temp = temp->next;
    } while (temp != head); // Continue until back to head (circular traversal)
    printf("Event '%s' not found.\n", eventName);
}

/* Removes a student with the given ID from a specific event */
void removeStudentFromEvent(const char* eventName, int studentId) {
    struct Event* temp = head;
    if (temp == NULL) {
        printf("No events available.\n");
        return;
    }
    do {
        if (strcmp(temp->eventName, eventName) == 0) {
            if (temp->students == NULL) {
                printf("No students registered for event '%s'.\n", eventName);
                return;
            }
            struct Student* curr = temp->students;
            struct Student* prev = NULL;
            while (curr != NULL) {
                if (curr->studentId == studentId) {
                    if (prev == NULL) {
                        temp->students = curr->next; // Remove head student
                    } else {
                        prev->next = curr->next; // Remove middle/last student
                    }
                    free(curr); // Free memory
                    printf("Student with ID %d removed from event '%s'.\n", studentId, eventName);
                    return;
                }
```

```
            prev = curr;
            curr = curr->next;
        }
        printf("Student with ID %d not found in event '%s'.\n", studentId, eventName);
        return;
    }
    temp = temp->next;
  } while (temp != head);
  printf("Event '%s' not found.\n", eventName);
}

/* Displays all events and their registered students */
void displayEvents() {
  if (head == NULL) {
    printf("No events registered.\n");
    return;
  }
  struct Event* temp = head;
  do {
    printf("\nEvent: %s\n", temp->eventName);
    printf("------------------------\n");
    if (temp->students == NULL) {
      printf("No students registered for this event.\n");
    } else {
      struct Student* studentTemp = temp->students;
      int count = 1;
      while (studentTemp != NULL) {
        printf("%d. ID: %d, Name: %s\n", count++, studentTemp->studentId, studentTemp->name);
        studentTemp = studentTemp->next;
      }
    }
```

```
        printf("------------------------\n");
        temp = temp->next;
    } while (temp != head); // Traverse the entire CLL
}

/* Frees all allocated memory to prevent memory leaks */
void freeMemory() {
    if (head == NULL) return;
    struct Event* currEvent = head;
    struct Event* nextEvent;
    do {
        struct Student* currStudent = currEvent->students;
        while (currStudent != NULL) {
            struct Student* temp = currStudent;
            currStudent = currStudent->next;
            free(temp); // Free each student node
        }
        nextEvent = currEvent->next;
        free(currEvent); // Free each event node
        currEvent = nextEvent;
    } while (currEvent != head);
}

/* Step 3: Main Function with Menu-Driven Interface */

int main() {
    int choice, studentId;
    char eventName[50], studentName[50];
    while (1) {
        printf("\n=== College Event Registration System ===\n");
        printf("1. Add Event\n");
```

```
printf("2. Add Student to Event\n");
printf("3. Remove Student from Event\n");
printf("4. Display All Events\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
    case 1:
        printf("Enter event name: ");
        scanf(" %[^\n]s", eventName);
        addEvent(eventName);
        break;
    case 2:
        printf("Enter event name: ");
        scanf(" %[^\n]s", eventName);
        printf("Enter student ID: ");
        scanf("%d", &studentId);
        printf("Enter student name: ");
        scanf(" %[^\n]s", studentName); // Fixed: Use studentName instead of eventName
        addStudentToEvent(eventName, studentId, studentName);
        break;
    case 3:
        printf("Enter event name: ");
        scanf(" %[^\n]s", eventName);
        printf("Enter student ID to remove: ");
        scanf("%d", &studentId);
        removeStudentFromEvent(eventName, studentId);
        break;
    case 4:
        displayEvents();
        break;
```

```
            case 5:
                freeMemory();
                printf("Exiting program. Memory freed.\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output:

=== College Event Registration System ===

1. Add Event

2. Add Student to Event

3. Remove Student from Event

4. Display All Events

5. Exit

Enter your choice: 1

Enter event name: Freshers Party

Event 'Freshers Party' added successfully.

Enter your choice: 2

Enter event name: Freshers Party

Enter student ID: 101

Enter student name: Ankit Sharma

Student 'Ankit Sharma' (ID: 101) added to event 'Freshers Party'.

Enter your choice: 2

Enter event name: Freshers Party

Enter student ID: 102

Enter student name: Priya Singh

Student 'Priya Singh' (ID: 102) added to event 'Freshers Party'.

Enter your choice: 4

Event: Freshers Party

---

1. ID: 101, Name: Ankit Sharma

2. ID: 102, Name: Priya Singh

Enter your choice: 3

Enter event name: Freshers Party

Enter student ID to remove: 101

Student with ID 101 removed from event 'Freshers Party'.

Enter your choice: 4

Event: Freshers Party

---

1. ID: 102, Name: Priya Singh

Enter your choice: 5

Exiting program. Memory freed.