EVENT MANAGEMENT SYSTEM

A MINI PROJECT FOR DSA LAB, SY COMP

PRESENTED BY:

SUMIT DESAI - 612415049
PIYUSH DESHPANDE - 612415052
BHAVESH GADLING - 612415061

Agenda

1. Introduction & Motivation
2. Module 1: Login Authentication
3. Module 2: Event Management
4. Module 3: Attendee Management
5. Data Structures Used
6. Time Complexity Highlights
7. Future Scope & Conclusion

Introduction

This project is a console-based Event Management System developed in C language. It allows organizers to create, modify, and manage events while attendees can easily register, unregister, and mark their attendance. The system supports smart venue booking with automatic time-clash detection, secure login/registration for both organizers and attendees, and stores all data permanently using CSV files. It uses efficient data structures such as Linked Lists, AVL Trees, Hash Tables, and Binary Search Trees to ensure fast searching, sorting, and management of events. Additional features include event search, chronological sorting, automatic removal of past events, and attendance statistics – making it a complete offline solution for managing college or club events effectively

Module 1: Login Authentication

- Features: Register | Login | Logout | Role (Organizer/Attendee)
- Data Stored: userAttendee.csv & userOrganizer.csv
- Key Structs:

1. struct user { int userId; char name[64]; ... };
2. struct userStatus { int userId; bool status; bool isOrg;

- Complexity: Login → O(n), Logout → O(1)
- Robust CSV parsing using strtok()

Module 2: Event Management

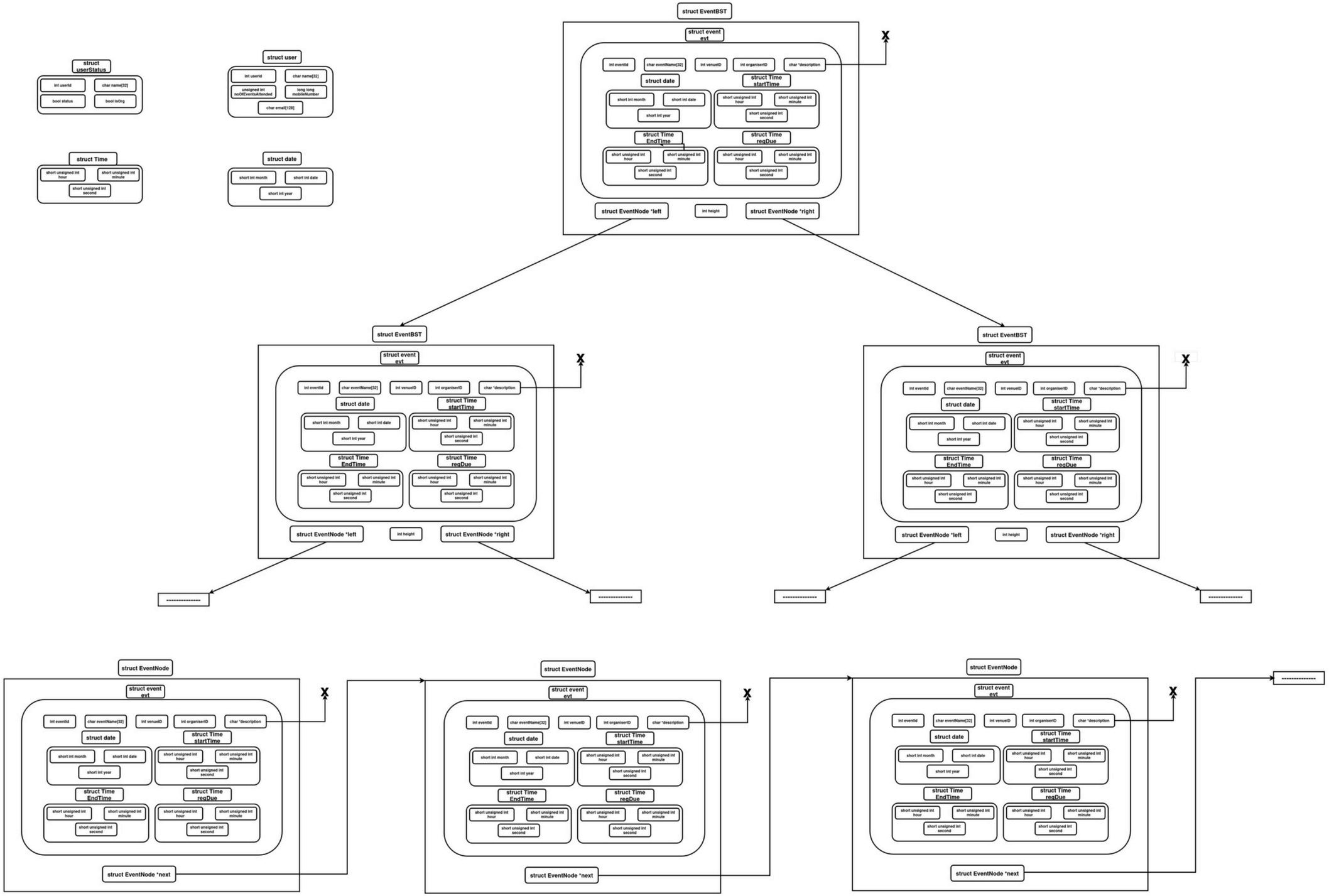
- Features: Add, Modify, Delete, Search, Sort Events
- Data Stored: events.csv, event_<eventID>.csv, organizer_<userID.csv>
- 4 Data Structures Used:

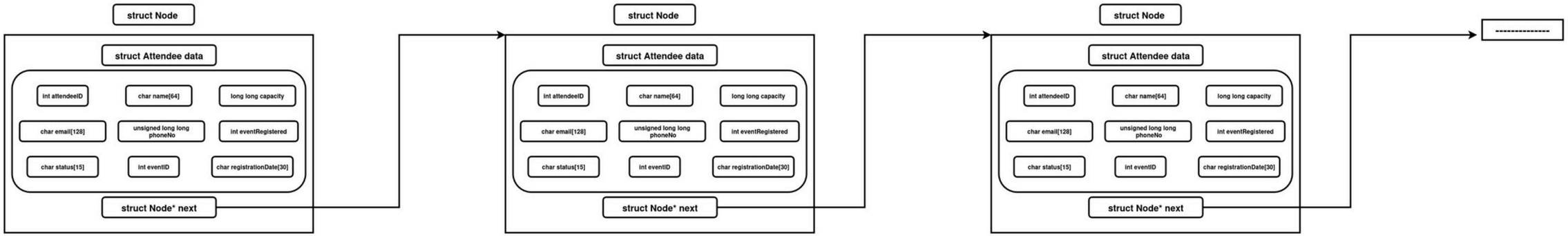
1. Singly Linked List → sequential access
2. AVL Tree → O(log n) search/delete by eventID
3. Hash Table (101 slots) → O(1) venue lookup
4. Interval Tree → overlap detection in O(m log m)

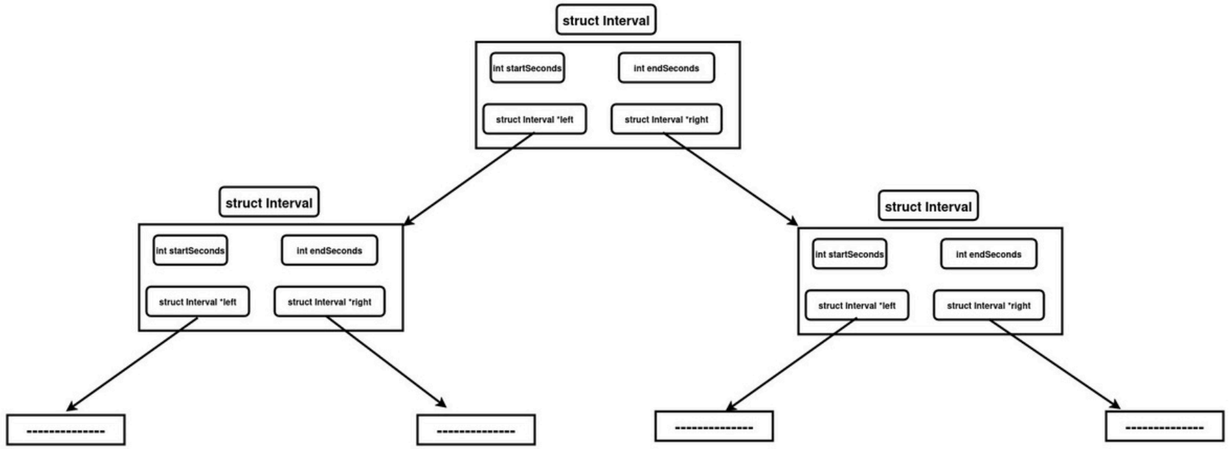
- QuickSort with function pointers for flexible sorting

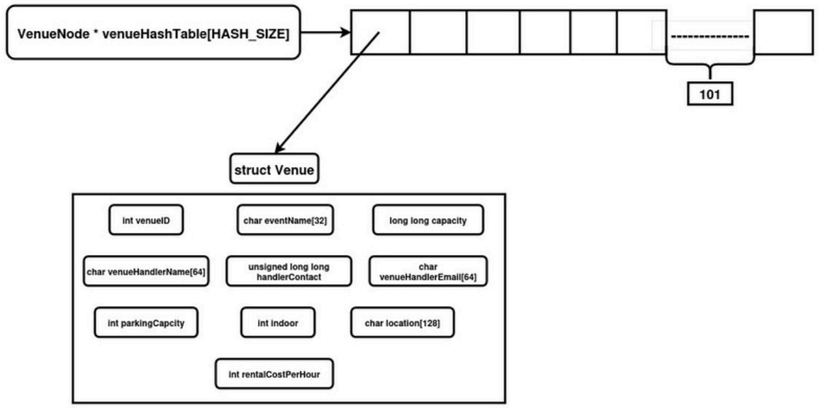
Module 3: Attendee Management

- Features: Register | Unregister | Mark Present | View Stats
- Data Stored: userAttendee.csv, event_<id>.csv
- Each event has its own linked list (event_5.csv)
- Auto-fetch user details from central database
- Key Functions:
  → registerAttendeeForEvent()
  → markAttendance()
  → viewStatistics()
  → shows attendance %
- All operations: O(n) → acceptable for real-world use

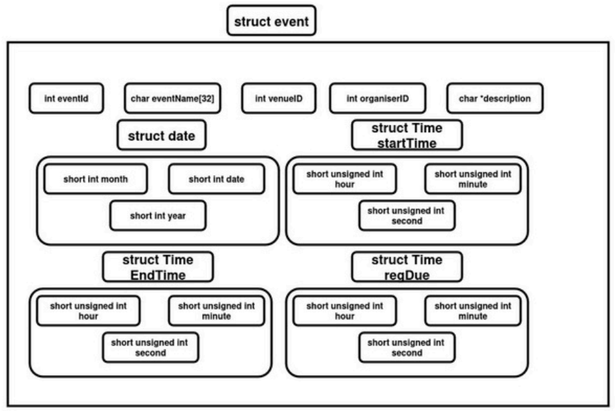

struct userStatus
int userId
char name[32]
bool status
bool isOrg
struct user
int userId
char name[32]
unsigned int noOfEventsAttended
long long mobileNumber
char email[128]
struct Time
short unsigned int hour
short unsigned int minute
short unsigned int second
struct date
short int month
short int date
short int year
struct EventBST
struct event evt
int eventId
char eventName[32]
int venueID
int organiserID
char *description
struct date
struct Time startTime
struct Time EndTime
struct Time regDue
struct EventNode *left
int height
struct EventNode *right
struct EventNode
struct EventNode *next

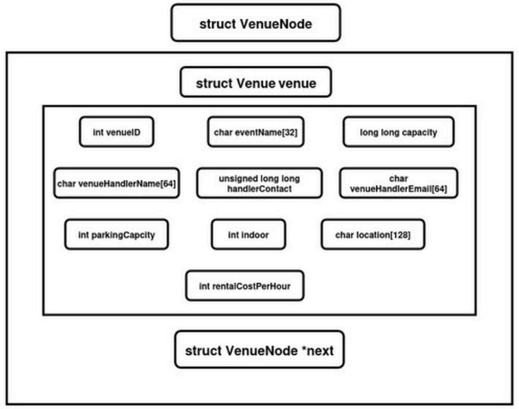

struct Node
struct Attendee data
int attendeeID
char name[64]
long long capacity
char email[128]
unsigned long long phoneNo
int eventRegistered
char status[15]
int eventID
char registrationDate[30]
struct Node* next
struct Node
struct Attendee data
int attendeeID
char name[64]
long long capacity
char email[128]
unsigned long long phoneNo
int eventRegistered
char status[15]
int eventID
char registrationDate[30]
struct Node* next
struct Node
struct Attendee data
int attendeeID
char name[64]
long long capacity
char email[128]
unsigned long long phoneNo
int eventRegistered
char status[15]
int eventID
char registrationDate[30]
struct Node* next

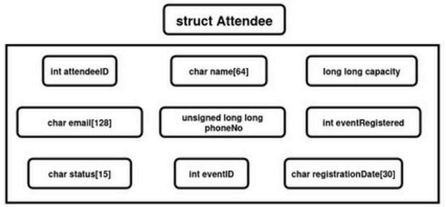

struct Interval
int startSeconds
int endSeconds
struct Interval *left
struct Interval *right
struct Interval
int startSeconds
int endSeconds
struct Interval *left
struct Interval *right
struct Interval
int startSeconds
int endSeconds
struct Interval *left
struct Interval *right

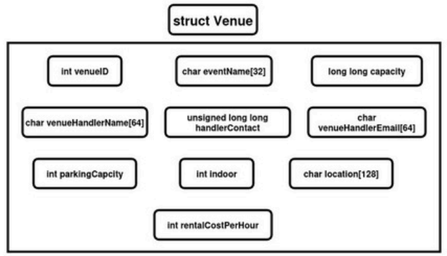

VenueNode * venueHashTable[HASH_SIZE]

101
struct Venue
int venueID
char eventName[32]
long long capacity
char venueHandlerName[64]
unsigned long long handlerContact
char venueHandlerEmail[64]
int parkingCapcity
int indoor
char location[128]
int rentalCostPerHour


struct event
int eventId
char eventName[32]
int venueID
int organiserID
char *description
struct date
short int month
short int date
short int year
struct Time startTime
short unsigned int hour
short unsigned int minute
short unsigned int second
struct Time EndTime
short unsigned int hour
short unsigned int minute
short unsigned int second
struct Time regDue
short unsigned int hour
short unsigned int minute
short unsigned int second


struct VenueNode
struct Venue venue
int venueID
char eventName[32]
long long capacity
char venueHandlerName[64]
unsigned long long handlerContact
char venueHandlerEmail[64]
int parkingCapcity
int indoor
char location[128]
int rentalCostPerHour
struct VenueNode *next


struct Attendee
int attendeeID
char name[64]
long long capacity
char email[128]
unsigned long long phoneNo
int eventRegistered
char status[15]
int eventID
char registrationDate[30]


struct Venue
int venueID
char eventName[32]
long long capacity
char venueHandlerName[64]
unsigned long long handlerContact
char venueHandlerEmail[64]
int parkingCapcity
int indoor
char location[128]
int rentalCostPerHour

Data Structures Summary

Sr. No.	Data Structure Used	Used In	Why Chosen	Complexity
1	Singly Linked List	Events & Attendees	Dynamic size, easy insert/delete	O(n) traversal
2	AVL Tree	Event ID lookup	Efficient storing and handling of events	O(log n)
3	Hash Table	Venue lookup	Average O(1) venue access	O(1) avg
4	Interval Tree	Time Conflict Check	Efficient event/venue overlap detection	O(m log m)

Time Complexity Highlights

- Slide 9 - Time Complexity Highlights
- Event Search/Delete → O(log n) [AVL]
- Venue Lookup → O(1) average [Hash Table]
- Venue Conflict Check → O(m log m) [Interval Tree]
- Sorting Events → O(n log n) [QuickSort]
- Attendee Operations → O(n) [Linked List] → Best possible theoretical bounds achieved

Future Scope

- Short Term:

1. Better console UI (colors, menus).
2. Serialize entire structures to binary files.

- Long Term:

1. Add GUI (using GTK or external lib)
2. Client-server model
3. Integrate SQLite database
4. Recommendation system using past data

Conclusion

- Successfully built a fully functional EMS in Pure C
- Applied 4+ advanced data structures meaningfully
- Achieved excellent time complexities
- Real-world applicable, scalable design
- Strong demonstration of DSA concepts
- Gained deep practical understanding of:
  → Balancing trade-offs in data structure selection
  → Importance of worst-case performance guarantees
  → File I/O challenges and safe parsing in C

Any Questions ?

THANK YOU!