WOLAITA SODO UNIVERSITY

SCHOOL OF INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

COURS TITLE: DATA STRUCTURE AND ALGORITHM

PROJECT TITLE: EVENT MANAGEMENT SYSTEM

GROUP 4

NO	NAME	ID
1	ASHENAFI DEGIF	UGR/91463/16
2	MAHLET AMSALU	UGR/91944/16
3	DEGNESH DALGA	UGR/91619/16
4	DAGIM DAWIT	UGR/92584/16
5	Akililu Abate	UGR/92526/16

Submitted to: Instructor

Submission date: 05/10/2017 E.C

ABSTRACT

The project is about Event Management. With the help of this project, each person can easily book various events like marriage, birthdays, anniversaries & so on. This system keeps the records of an event & effectively manages all the information related to the various events that take place in an organization.

One can easily book an event manually within their budget, look over the available dates, can deposit for the booked event.

This system allows paying for the bills without going to a bank. Hence, it helps to choose the preferable event effortlessly.

The primary purpose of this system is to provide a wide variety of services with zero workforces & less time-consuming.

There is detailed information about the owners with contact information. The design provides a secure, error-free, reliable & fast management system & ensure user have better utilization of given resources. In short, it is a convenient & flexible system for managing an event with detailed information.

INTRODUCTION

In today's world, organizing events such as seminars, workshops, conferences, and concerts has become increasingly common. Managing registrations for such events requires a system that can handle a large number of attendees efficiently and fairly.

Our Event Management System is designed to address this need. It allows users to:

View event details

Register for events

Book seats (in future versions)

Have their registrations processed in the order received

This project demonstrates our understanding of data structures, especially queues, and their use in building real-world applications. The application simulates event registration logic with a first-come, first-served approach using the queue data structure.

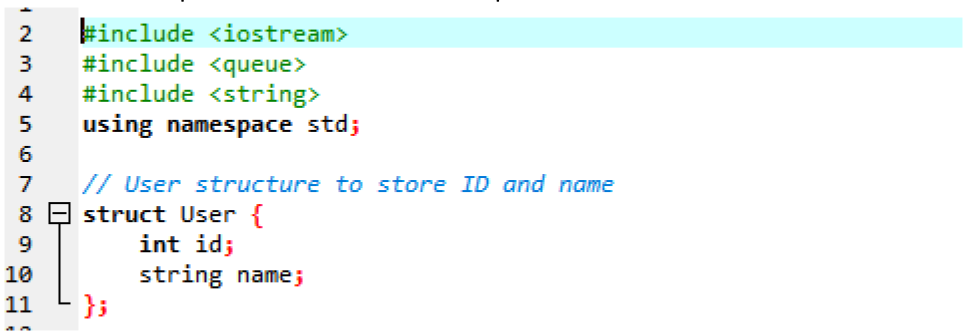
PROBLEM STATEMENT

The objective of this project is to build an event management system that helps organize and manage events by users to register, book seats, and check event details. The event uses queues as the core data structure to handle the event register efficiently in the first-come, first-served manager(FIFO).

DESIGN AND IMPEMENTATION

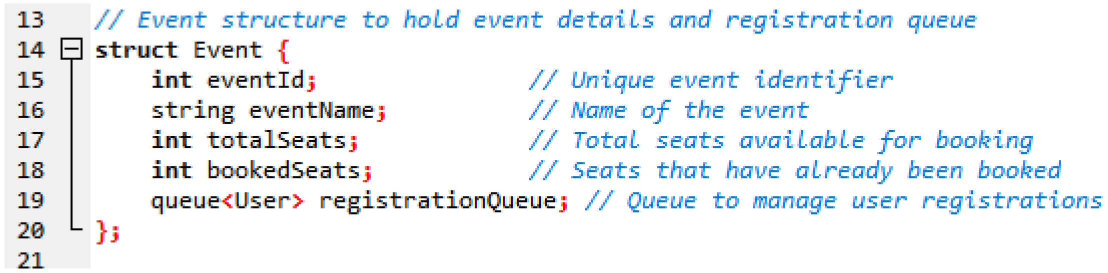
Data structure used;

- ✓ Struct user: Represents a user with a unique ID and name.



```cpp
#include <iostream>
#include <queue>
#include <string>
using namespace std;

// User structure to store ID and name
struct User {
    int id;
    string name;
};
```

- ✓ Struct event: represents an event with an ID, name, total seats, and a queue of users.

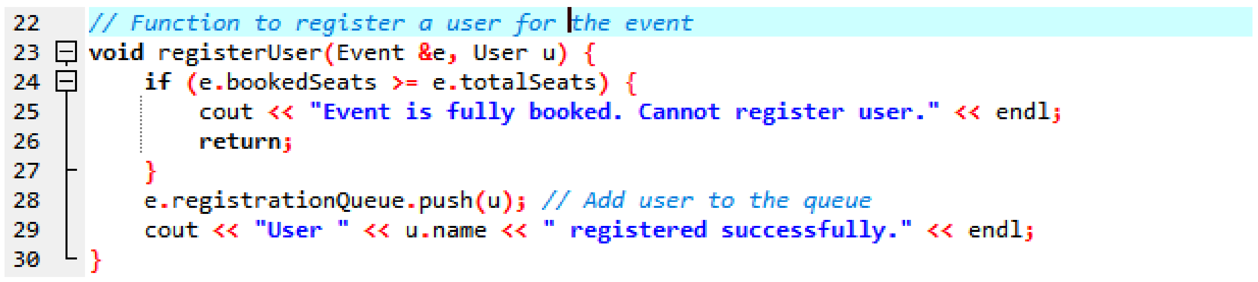


```cpp
// Event structure to hold event details and registration queue
struct Event {
    int eventId;                  // Unique event identifier
    string eventName;             // Name of the event
    int totalSeats;               // Total seats available for booking
    int bookedSeats;              // Seats that have already been booked
    queue<User> registrationQueue; // Queue to manage user registrations
};

```

- ✓ Queue<User>: Manages the order of user registrations.

Functions implemented:

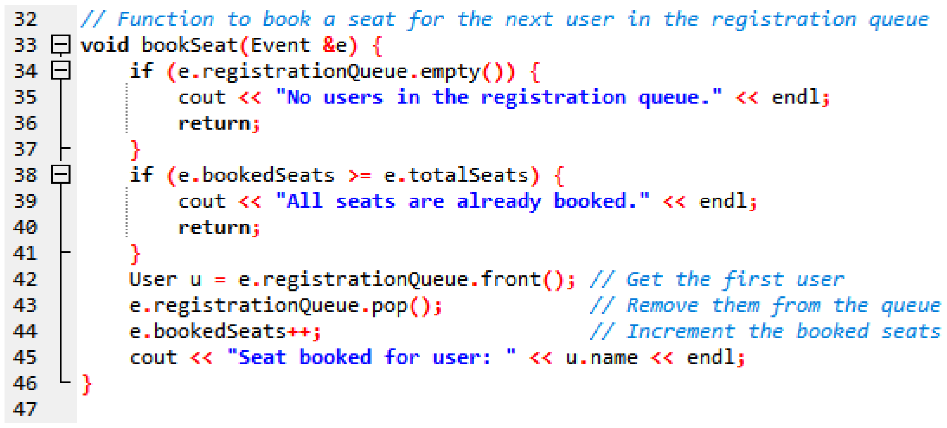
- ✓ registerUser(event &e, User U):
  Adds a user to the event's registration queue if seats are available.



```cpp
// Function to register a user for the event
void registerUser(Event &e, User u) {
    if (e.bookedSeats >= e.totalSeats) {
        cout << "Event is fully booked. Cannot register user." << endl;
        return;
    }
    e.registrationQueue.push(u); // Add user to the queue
    cout << "User " << u.name << " registered successfully." << endl;
}
```

- ✓ bookSeat(Event &e):

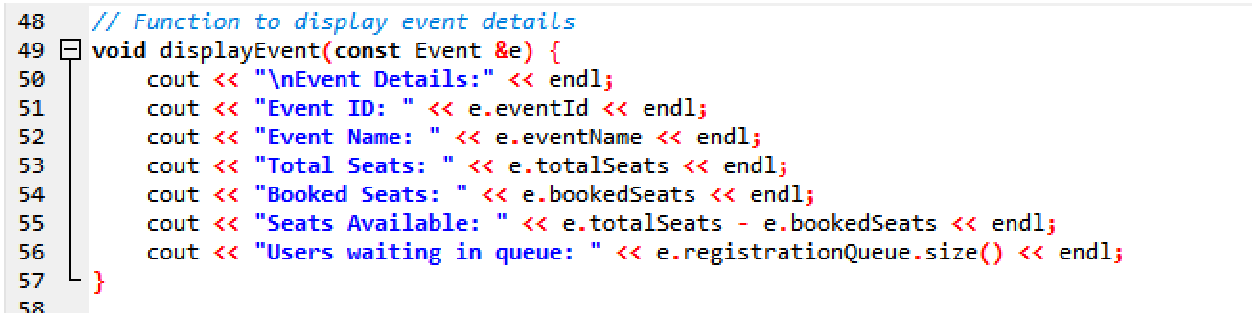
  Books a seat for the next user in the queue.



```cpp
// Function to book a seat for the next user in the registration queue
void bookSeat(Event &e) {
    if (e.registrationQueue.empty()) {
        cout << "No users in the registration queue." << endl;
        return;
    }
    if (e.bookedSeats >= e.totalSeats) {
        cout << "All seats are already booked." << endl;
        return;
    }
    User u = e.registrationQueue.front(); // Get the first user
    e.registrationQueue.pop();            // Remove them from the queue
    e.bookedSeats++;                      // Increment the booked seats
    cout << "Seat booked for user: " << u.name << endl;
}
```

- ✓ displayEvent(const Event &e):

  Displays details of the event including number of seats and queue status.



```cpp
// Function to display event details
void displayEvent(const Event &e) {
    cout << "\nEvent Details:" << endl;
    cout << "Event ID: " << e.eventId << endl;
    cout << "Event Name: " << e.eventName << endl;
    cout << "Total Seats: " << e.totalSeats << endl;
    cout << "Booked Seats: " << e.bookedSeats << endl;
    cout << "Seats Available: " << e.totalSeats - e.bookedSeats << endl;
    cout << "Users waiting in queue: " << e.registrationQueue.size() << endl;
}
```

Coding practices:

- ✓ Modular functions
- ✓ Clear naming conventions
- ✓ Basic input validation and status messages

ALGORITHM ANALYSIS

Time complexity:

✓RegisterUser O(1): Adds a user to the queue.

✓bookSeat O(1): Removes the next user from the queue.

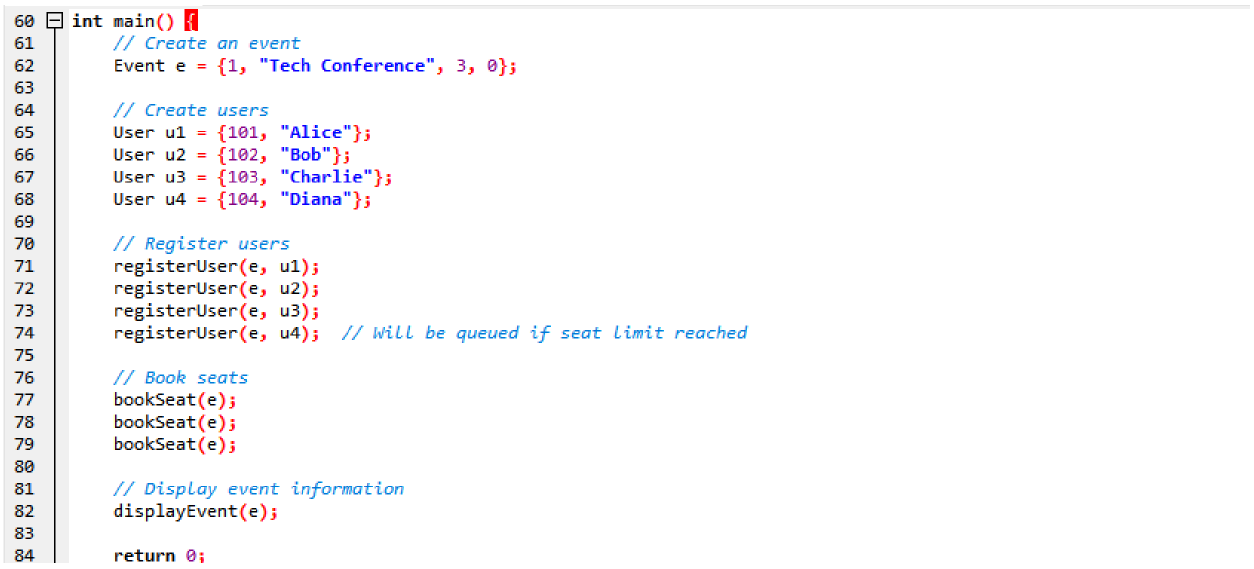
✓displayEvent O(1): Constant-time printing.

Space complexity:

✓O(n): Where n is the number of registered users in the queue.

TESTING AND PERFORMANCE ANALYSIS

Test case:

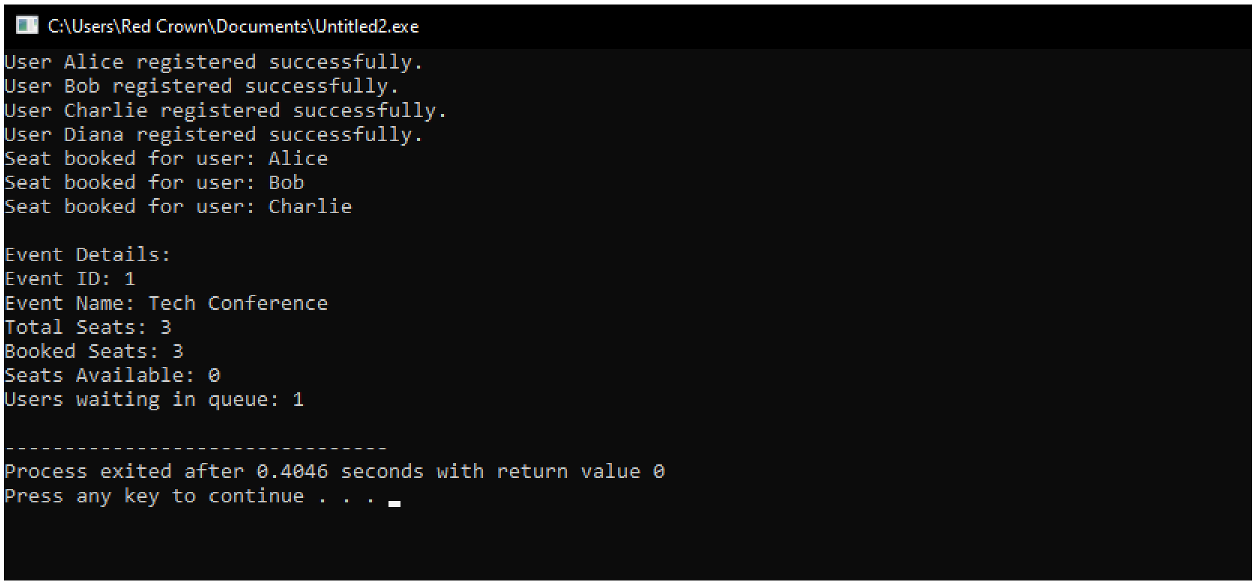


```
int main() {
    // Create an event
    Event e = {1, "Tech Conference", 3, 0};

    // Create users
    User u1 = {101, "Alice"};
    User u2 = {102, "Bob"};
    User u3 = {103, "Charlie"};
    User u4 = {104, "Diana"};

    // Register users
    registerUser(e, u1);
    registerUser(e, u2);
    registerUser(e, u3);
    registerUser(e, u4);  // Will be queued if seat limit reached

    // Book seats
    bookSeat(e);
    bookSeat(e);
    bookSeat(e);

    // Display event information
    displayEvent(e);

    return 0;
```

✓ Created one event with three seats.

✓ Registered four users.

- ✓ Booked seats for the first three users.
- ✓ Displayed event details including remaining users in the queue.



```
C:\Users\Red Crown\Documents\Untitled2.exe
User Alice registered successfully.
User Bob registered successfully.
User Charlie registered successfully.
User Diana registered successfully.
Seat booked for user: Alice
Seat booked for user: Bob
Seat booked for user: Charlie

Event Details:
Event ID: 1
Event Name: Tech Conference
Total Seats: 3
Booked Seats: 3
Seats Available: 0
Users waiting in queue: 1

--------------------------------
Process exited after 0.4046 seconds with return value 0
Press any key to continue . . .
```

Performance observations:

- ✓ All queue operations performed in constant time.
- ✓ Correct users were served in order of registration.
- ✓ Prevented overbooking of seats.

<u>DOCUMENTATION</u>

How to Use:

- ✓ Compile and run C++ program.
- ✓ Users are hardcoded; system can be extended to take user input.

- ✓ Observe registration, booking, and event status output on console.

Usage summary:

- ✓ registerUser() is used to simulate user registration.
- ✓ bookSeat() processes booking for the next in line.
- ✓ displayEvent() shows real-time event status.

PRESENTATION SUMMARY

- ✓ Introduced the problem and goal of the system.
- ✓ Explained choice of queue for fairness.
- ✓ Demonstrated core features: registration, booking, viewing status.
- ✓ Analyzed algorithm performance and tested behaviour.
- ✓ Discussed project strengths and limitations.

Strengths:

- ✓ Realistic simulation of an event queue.
- ✓ Efficient and fair processing.
- ✓ Simple and readable code.

Limitations:

- ✓ Only supports a single event.
- ✓ No dynamic user input or persistent.

Future Improvements

- ✓ Support for multiple events.
- ✓ Store event and user data using file or database.
- ✓ Add login system for admin and users.
- ✓ Allow real-time seat allocation.
- ✓ Build a simple web or GUI interface.

CONCLUSION

This project successfully applies data structures and algorithm to build a practical Event Management System. The use of queues to enables fair user registration and seat booking. The system demonstrates key data structure and algorithm concepts such as modular design, space/time complexity analysis, and efficient algorithm implementation.

-----Thankyou-----