#### PROJECT WORK OF

#### DATA STRUCTURE LAB USING C….

#### C:\Users\HP\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\D1A21DA7BCA4ABFF8B0B61B87597DE73\WhatsApp Image 2024-11-05 at 19.35.44_a2cef127.jpg

**\*Event Ticket Booking System using Queue, Linked List, and Tree in C\***

**\*Abstract\***

The Event Ticket Booking System is a C-based project that manages event ticket reservations efficiently using fundamental data structures: Queue, Linked List, and Tree. The system provides features such as ticket booking, cancellation, and event management. The Queue is used for managing booking requests, the Linked List handles the list of attendees, and a Tree structure organizes event categories or seating arrangements. This project aims to demonstrate efficient data handling and management using these structures.

**\*Project Report\***

**\*Introduction\***

Event management requires a structured way to handle ticket reservations, cancellations, and event organization. This project implements a ticket booking system utilizing Queue, Linked List, and Tree to ensure efficient data management.

**\*Features\***

1. \*Event Management\* - Organize events using a Tree structure.

2. \*Ticket Booking\* - Uses a Queue to manage booking requests.

3. \*Attendee List\* - Maintains attendee records using a Linked List.

4. \*Cancellation\* - Allows ticket cancellation with real-time updates.

5. \*Display & Search\* - Shows available events, booked tickets, and search functionality.

**\*Data Structure Usage\***

- \*Queue\*: Used for processing ticket booking requests in a first-come, first-served manner.

- \*Linked List\*: Manages attendees and their ticket details dynamically.

- \*Tree\*: Organizes events based on categories or venues for efficient searching and management.

**\*Project Code\***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// ---- Queue (for booking requests) ----

#define MAX 100

char queue[MAX][50];

int front = -1, rear = -1;

void enqueue(char customer\_name[]) {

if (rear == MAX - 1) {

printf("Queue is full. Cannot add more requests.\n");

} else {

rear++;

strcpy(queue[rear], customer\_name);

if (front == -1)

front = 0;

printf("Customer %s added to the booking queue.\n", customer\_name);

}

}

void dequeue() {

if (front == -1 || front > rear) {

printf("Queue is empty. No requests to process.\n");

} else {

printf("Processing booking for %s.\n", queue[front]);

front++;

}

}

// ---- Linked List (for managing booked tickets) ----

struct Node {

char customer[50];

struct Node\* next;

};

struct Node\* global\_head = NULL; // Global linked list for bookings

void add\_global\_booking(char customer[], int event\_id) {

struct Node\* new\_booking = (struct Node\*)malloc(sizeof(struct Node));

strcpy(new\_booking->customer, customer);

new\_booking->next = global\_head;

global\_head = new\_booking;

printf("Customer %s booked successfully for Event ID: %d.\n", customer, event\_id);

}

// ---- Binary Search Tree (for managing events) ----

struct Event {

int id;

char name[50];

int available\_seats;

struct Event\* left;

struct Event\* right;

struct Node\* bookings; // Linked list for this event's bookings

};

struct Event\* root = NULL;

struct Event\* insert\_event(struct Event\* node, int id, char name[], int seats) {

if (node == NULL) {

struct Event\* new\_event = (struct Event\*)malloc(sizeof(struct Event));

new\_event->id = id;

strcpy(new\_event->name, name);

new\_event->available\_seats = seats;

new\_event->left = new\_event->right = NULL;

new\_event->bookings = NULL; // Initialize bookings

return new\_event;

}

if (id < node->id)

node->left = insert\_event(node->left, id, name, seats);

else if (id > node->id)

node->right = insert\_event(node->right, id, name, seats);

return node;

}

void display\_events(struct Event\* node) {

if (node != NULL) {

display\_events(node->left);

printf("Event ID: %d | Name: %s | Available Seats: %d\n", node->id, node->name, node->available\_seats);

display\_events(node->right);

}

}

void add\_customer\_to\_event(struct Event\* node, int event\_id, char customer\_name[]) {

if (node == NULL) {

printf("Event not found.\n");

return;

}

if (event\_id == node->id) {

if (node->available\_seats > 0) {

struct Node\* new\_customer = (struct Node\*)malloc(sizeof(struct Node));

strcpy(new\_customer->customer, customer\_name);

new\_customer->next = node->bookings;

node->bookings = new\_customer;

node->available\_seats--; // Decrease available seats

enqueue(customer\_name); // Add the customer to the queue

add\_global\_booking(customer\_name, event\_id);

printf("Customer %s added to Event: %s.\n", customer\_name, node->name);

} else {

printf("No seats available for this event.\n");

}

return;

} else if (event\_id < node->id) {

add\_customer\_to\_event(node->left, event\_id, customer\_name);

} else {

add\_customer\_to\_event(node->right, event\_id, customer\_name);

}

}

void cancel\_booking(struct Event\* node, int event\_id, char customer\_name[]) {

if (node == NULL) {

printf("Event not found.\n");

return;

}

if (event\_id == node->id) {

struct Node\* current = node->bookings;

struct Node\* prev = NULL;

while (current != NULL && strcmp(current->customer, customer\_name) != 0) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Booking not found for %s in Event: %s.\n", customer\_name, node->name);

return;

}

// Remove the booking

if (prev == NULL) {

node->bookings = current->next;

} else {

prev->next = current->next;

}

free(current);

node->available\_seats++; // Increase available seats

printf("Booking canceled for %s in Event: %s.\n", customer\_name, node->name);

return;

} else if (event\_id < node->id) {

cancel\_booking(node->left, event\_id, customer\_name);

} else {

cancel\_booking(node->right, event\_id, customer\_name);

}

}

void display\_available\_seats(struct Event\* node) {

if (node != NULL) {

display\_available\_seats(node->left);

printf("Event: %s | Event ID: %d | Available Seats: %d\n", node->name, node->id, node->available\_seats);

display\_available\_seats(node->right);

}

}

// ------------------- MAIN FUNCTION -------------------

int main() {

int choice, event\_id;

char customer\_name[50], event\_name[50];

// Adding some sample events

root = insert\_event(root, 1, "Rock Concert", 100);

root = insert\_event(root, 2, "Tech Conference", 150);

root = insert\_event(root, 3, "Art Exhibition", 200);

while (1) {

printf("\n===== Event Ticket Booking System =====\n");

printf("1. View All Events\n");

printf("2. Add Customer to Event\n");

printf("3. Cancel Booking\n");

printf("4. Display Current Available Seats\n");

printf("5. Process Queue\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // Clear buffer

switch (choice) {

case 1:

printf("Available Events:\n");

display\_events(root);

break;

case 2:

printf("Enter Event ID to add customer to: ");

scanf("%d", &event\_id);

getchar(); // Clear buffer

printf("Enter Customer Name: ");

fgets(customer\_name, sizeof(customer\_name), stdin);

customer\_name[strcspn(customer\_name, "\n")] = 0;

add\_customer\_to\_event(root, event\_id, customer\_name);

break;

case 3:

printf("Enter Event ID to cancel booking from: ");

scanf("%d", &event\_id);

getchar(); // Clear buffer

printf("Enter Customer Name to cancel booking: ");

fgets(customer\_name, sizeof(customer\_name), stdin);

customer\_name[strcspn(customer\_name, "\n")] = 0;

cancel\_booking(root, event\_id, customer\_name);

break;

case 4:

printf("Current Available Seats:\n");

display\_available\_seats(root);

break;

case 5:

dequeue();

break;

case 6:

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

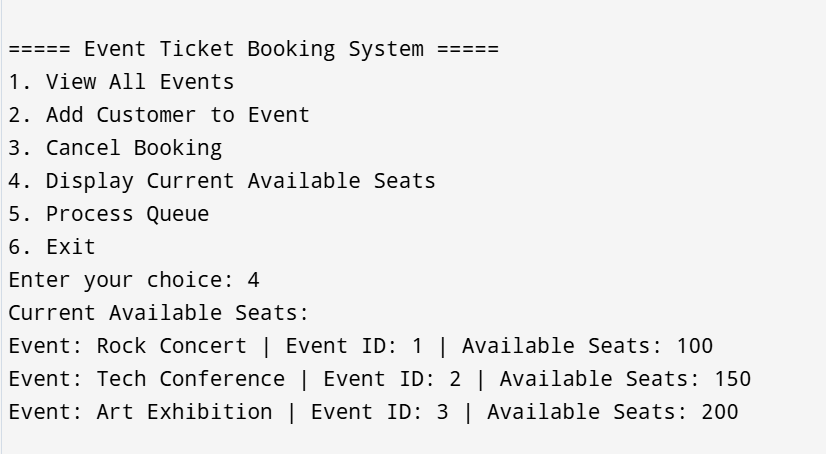
}

return 0;

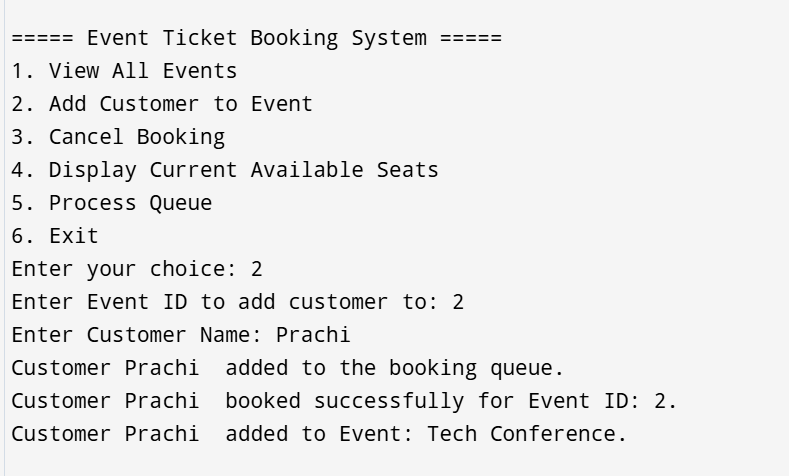
}

**\*Output Screenshots\***

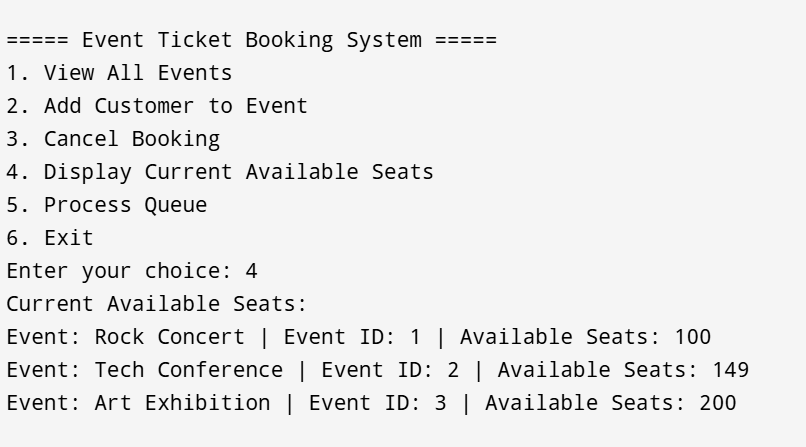
* **VIEW ALL EVENTS .....**

****

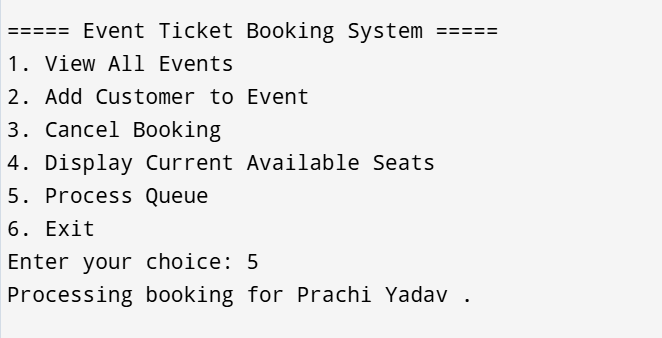
* **ADD CUSTOMER TO EVENT ….**

****

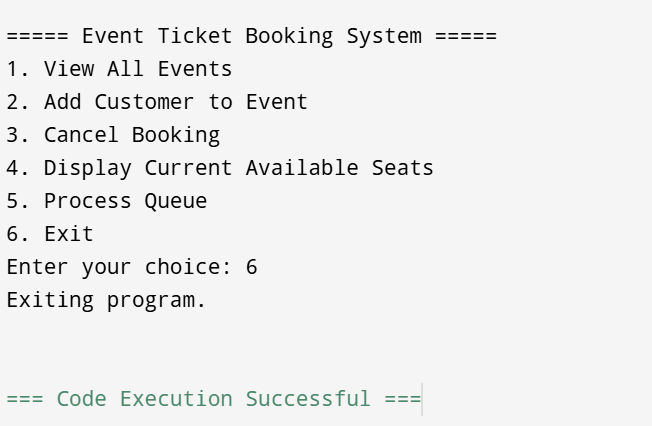
* **TO CHECK CURRENT AVAILABLE SEATS.....**

****

* **PROCESS QUEUE....**



* **TO EXIT PROGRAM ....**



**\*EXPLANATION OF THE CODE\***

1. \*Booking a Ticket:\* Displays the queueing of ticket requests.

2. \*Displaying Attendees:\* Lists attendees who have booked tickets.

3. \*Event Categories:\* Shows a structured tree of events.

4. \*Cancelling a Ticket:\* Demonstrates ticket removal from the list.

**\*Conclusion\***

This project successfully implements an Event Ticket Booking System using Queue, Linked List, and Tree in C. It demonstrates efficient data handling and event management, making it an ideal system for small-scale event organizers.

***\*THANK YOU\****