**Abstract**

This code is an abstract implementation of a movie ticket booking system using C programming language. Here's a breakdown of the different parts of the code:

1. \*Initialization:\*

- Set up the available movies, showtimes, theater layout, and seat availability.

2. \*User Interaction:\*

- Display a list of available movies and showtimes.

- Allow the user to select a movie and showtime.

3. \*Seat Selection:\*

- Display the theater layout with available and booked seats.

- Allow the user to select seats based on availability.

4. \*Payment:\*

- Calculate the total cost based on the number of selected seats and movie price.

- Collect payment information securely.

5. \*Confirmation:\*

- Display a summary of the booking for user confirmation.

6. \*Updating Seat Availability:\*

- Mark the selected seats as booked to prevent double booking.

7. \*Data Management:\*

- Maintain databases for movies, showtimes, bookings, and seat availability.

8. \*Error Handling:\*

- Handle scenarios like payment failures, seat unavailability, or network errors gracefully.

9. \*Cancellation and Refunds:\*

- Allow users to cancel bookings within a certain time frame for a partial or full refund.

- Update seat availability accordingly.

10. \*Reporting and Analytics:\*

- Generate reports on daily bookings, popular movies, and occupancy rates.

**Objectives**

. Here are the main objectives of this code:

1. \*Define Data Structures:\*

- Define a structure `struct Movie` to store information about a movie, including its ID, name, and available seats.

- Define a structure `struct Customer` to store information about a customer, including their ID, name, and the number of tickets they want.

2. \*Manage Movies:\*

- Implement a linked list structure for movies (`struct Movie`) to keep track of available movies, each with their own details such as ID, name, and available seats.

- Provide a function `addMovie` to add movies to the linked list of movies, initializing their attributes.

3. \*Manage Booking Requests:\*

- Implement a queue structure for booking requests (`struct Customer`) to hold customer information and their requested number of tickets.

- Provide a function `enqueueBookingRequest` to enqueue booking requests, adding customers to the end of the booking queue.

4. \*Process Booking Requests:\*

- Implement a function `processBookingRequests` to process booking requests from the queue.

- For each booking request, search through the movie list to find a movie with enough available seats.

- If a suitable movie is found, decrease its available seats by the requested number of tickets and print a booking success message.

- If no suitable movie is found, print a booking failure message indicating unavailability of seats.

5. \*Main Function:\*

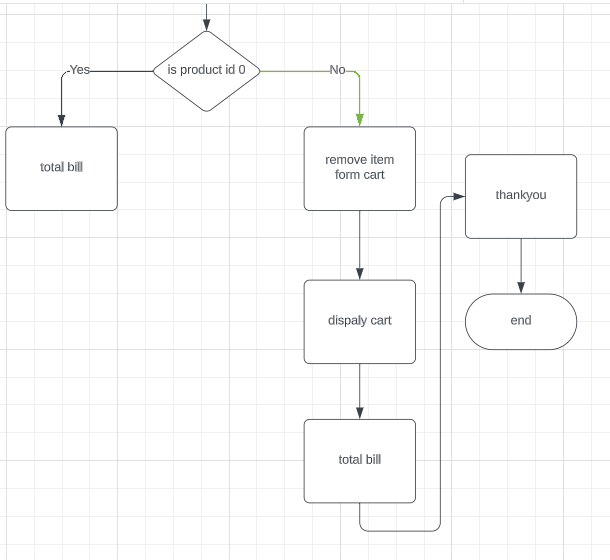
- In the `main` function, populate the movie list using the `addMovie` function with initial movie details.

- Enqueue booking requests using the `enqueueBookingRequest` function with customer details and requested ticket counts.

- Process the booking requests using the `processBookingRequests` function, which checks seat availability and provides booking outcomes.

Overall, the code aims to simulate a simple ticket booking system where movies are listed with available seats and customers can request tickets. It demonstrates the use of linked lists for movies and queues for booking requests, highlighting the basic flow of adding movies, enqueuing booking requests, and processing those requests based on seat availability. However, this code is quite basic and lacks error handling and more advanced features that would be found in a real-world ticket booking system.

**Flow chart**

****

**Explanation of code**

This code implements a basic movie ticket booking system using C programming. Here's an explanation of the code step by step:

1. The code starts by including necessary header files: `<stdio.h>` for standard input/output functions and `<stdlib.h>` for memory allocation functions.

2. Two structures, `struct Movie` and `struct Customer`, are defined to represent movies and customers, respectively. Each structure has attributes like ID, name, available seats, and a pointer to the next structure of the same type (linked list).

3. Two global pointers, `movieList` and `queueFront`, are initialized to `NULL` to serve as the heads of linked lists representing movies and booking requests (queue), respectively. `queueRear` is another pointer to keep track of the rear of the queue.

4. The `addMovie` function is defined to add a movie to the movie list. It creates a new `struct Movie` node, initializes its attributes, and adds it to the linked list of movies.

5. The `enqueueBookingRequest` function is defined to enqueue a booking request into the booking queue. It creates a new `struct Customer` node, initializes its attributes, and adds it to the end of the queue.

6. The `processBookingRequests` function processes booking requests from the queue. It iterates through the queue, and for each customer, it searches for a movie with enough available seats. If a suitable movie is found, it decrements the available seats and prints a success message. If no suitable movie is found, it prints a failure message. After processing, the customer node is dequeued and memory is freed.

7. In the `main` function:

- Movies are added to the movie list using the `addMovie` function.

- Booking requests are enqueued using the `enqueueBookingRequest` function.

- Booking requests are then processed using the `processBookingRequests` function.

8. The program concludes by returning 0, indicating successful execution.

This code represents a simplified ticket booking system where movies are stored in a linked list, and booking requests are managed in a queue. However, it lacks error handling, input validation, and more advanced features that a real-world ticket booking system might need.

**Implementation of code**

#include <stdio.h>

#include <stdlib.h>

// Structure for a movie

struct Movie {

int id;

char name[50];

int availableSeats;

struct Movie \*next;

};

// Structure for a customer

struct Customer {

int id;

char name[50];

int numTickets;

struct Customer \*next;

};

// Linked list for movies

struct Movie \*movieList = NULL;

// Queue for booking requests

struct Customer \*queueFront = NULL;

struct Customer \*queueRear = NULL;

// Function to add a movie to the list

void addMovie(int id, const char \*name, int availableSeats) {

// Create a new movie node

struct Movie \*newMovie = (struct Movie \*)malloc(sizeof(struct Movie));

newMovie->id = id;

strcpy(newMovie->name, name);

newMovie->availableSeats = availableSeats;

newMovie->next = NULL;

// Add the movie to the list

if (movieList == NULL) {

movieList = newMovie;

} else {

struct Movie \*temp = movieList;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newMovie;

}

}

// Function to enqueue a booking request

void enqueueBookingRequest(int customerId, const char \*customerName, int numTickets) {

// Create a new customer node

struct Customer \*newCustomer = (struct Customer \*)malloc(sizeof(struct Customer));

newCustomer->id = customerId;

strcpy(newCustomer->name, customerName);

newCustomer->numTickets = numTickets;

newCustomer->next = NULL;

// Enqueue the customer

if (queueFront == NULL) {

queueFront = newCustomer;

queueRear = newCustomer;

} else {

queueRear->next = newCustomer;

queueRear = newCustomer;

}

}

// Function to process booking requests

void processBookingRequests() {

// Process booking requests in the queue

while (queueFront != NULL) {

struct Customer \*customer = queueFront;

// Find the movie with available seats

struct Movie \*temp = movieList;

while (temp != NULL) {

if (temp->availableSeats >= customer->numTickets) {

temp->availableSeats -= customer->numTickets;

printf("Booking successful for %s\n", customer->name);

break;

}

temp = temp->next;

}

if (temp == NULL) {

printf("Booking failed for %s due to unavailability of seats\n", customer->name);

}

queueFront = queueFront->next;

free(customer);

}

}

int main() {

// Add movies to the list

addMovie(1, "Movie A", 50);

addMovie(2, "Movie B", 40);

// Add more movies...

// Enqueue booking requests

enqueueBookingRequest(1, "Customer X", 2);

enqueueBookingRequest(2, "Customer Y", 4);

// Enqueue more requests...

// Process booking requests

processBookingRequests();

return 0;

}

**Output**

Online

Shopping