EVENT TICKETING SYSTEM

Name: Tony Antony
Roll no: 71
Course: Computer
Date : 17/07/24

INTRODUCTION

BRIEF OVERVIEW

The Event Ticketing System is a console-based application designed to manage events and ticket sales efficiently. Developed in C, this program allows users to add new events, purchase tickets for existing events, and view a list of all events with their details. The system ensures data persistence using file operations to store event information and handles various scenarios, such as checking for expired events and preventing duplicate event IDs. This application provides a straightforward and interactive way to handle basic event management and ticketing functions.

PROBLEM STATEMENT

Managing event ticket sales manually is inefficient and prone to errors

Solution: The Event Ticketing System automates the process of adding events, purchasing tickets, and viewing event details, ensuring accurate and efficient management.

Key Features:- Add new events with unique IDs.

- Purchase tickets while checking event validity.
- View all event details in a structured format.

OBJECTIVE

The objective of the Event Ticketing System is to automate the process of managing event ticket sales, ensuring efficient, accurate, and user-friendly handling of event creation, ticket purchasing, and event information display. This system aims to eliminate the errors and inefficiencies associated with manual ticketing processes by providing a robust, file-based solution for storing and retrieving event data.

SYSTEM REQUIREMENTS

Software:- C compiler (e.g., GCC)

- Operating System: Windows, macOS, or Linux
- Text editor or IDE (e.g., Visual Studio Code, Code::Blocks)

Hardware:- Minimum 1 GHz processor

- 512 MB RAM
- 50 MB available disk space
- Monitor and keyboard for input and output

DESIGN AND DEVELOPMENT

Design

1. **Data Structure**: Use a struct to define an Event with fields for ID, name, total tickets, sold tickets, remaining tickets, and event date.
2. **File Operations:** Store event data in a binary file (events.dat) for persistence. Implement functions for reading, writing, and updating the file.
3. **User Interface**: Provide a console-based menu for user interaction, offering options to add events, purchase tickets, and view events

DESIGN AND DEVELOPMENT

Development

1: **Language:** Develop the system in C.

2. **Modules:** - addevent(): Add a new event, checking for duplicate IDs and ensuring proper input.
   - purchaseticket(): Handle ticket purchases, ensuring event validity and ticket availability.
   - viewevents(): Display a list of all events with their details.
   - menu(): Present the main menu and handle user choices.

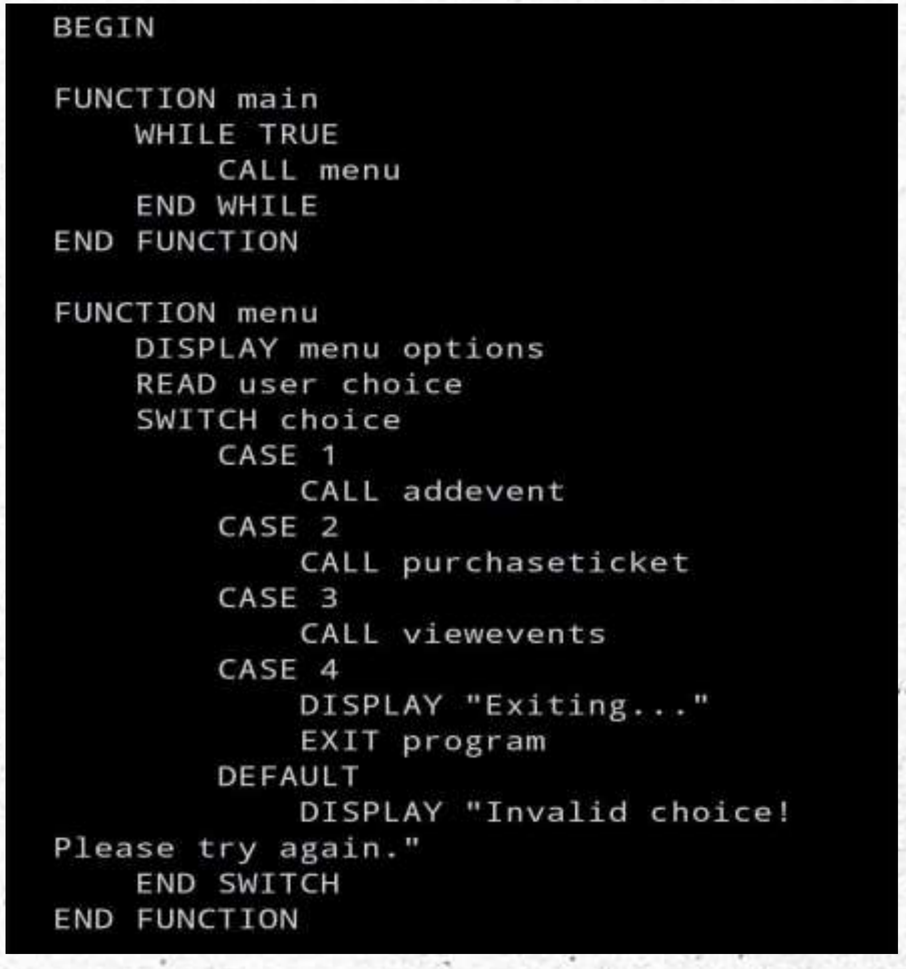
3. **Error Handling:** Implement error checks for file operations and user inputs

4. **Testing:** Conduct thorough testing for each function to ensure correct operation and handle edge cases, such as invalid dates and full events.

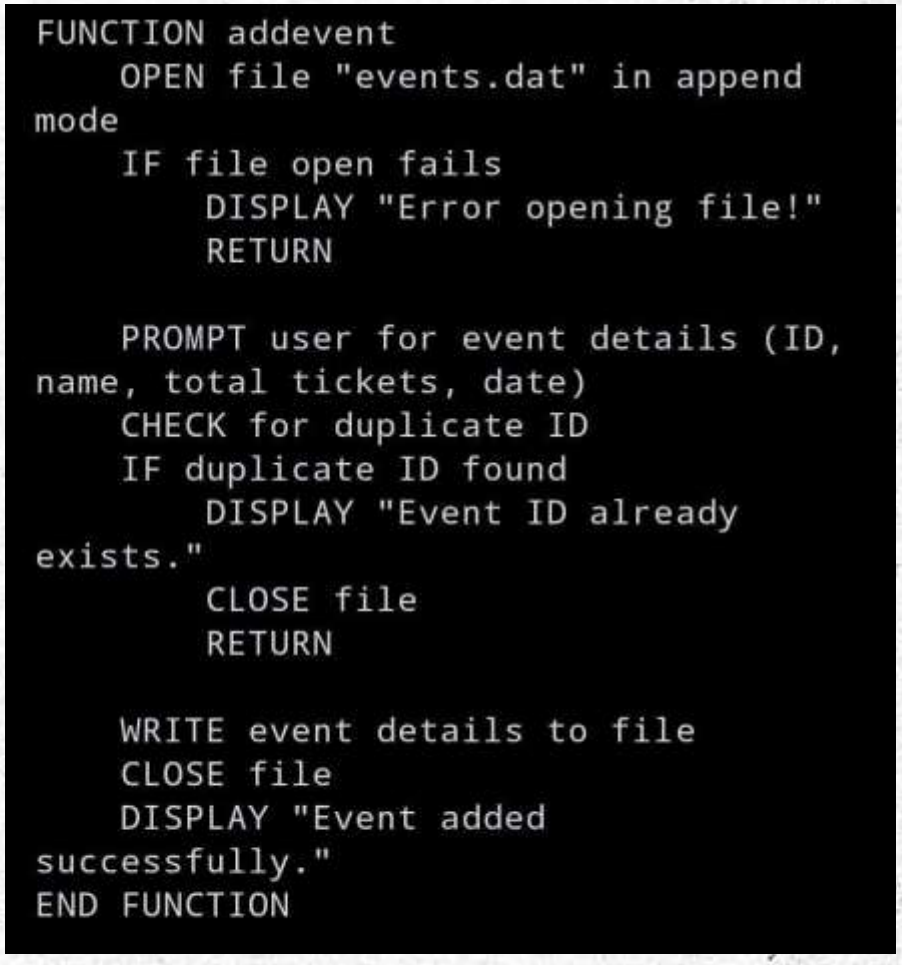
PROGRAM LOGIC

The Event Ticketing System is a console-based program written in C, structured around a main menu that provides options to add events, purchase tickets, and view events. The system uses a struct to define event properties and stores event data in a binary file (events.dat). When adding an event, the program checks for duplicate IDs and ensures valid input. Ticket purchases are validated against event dates and ticket availability, updating the event data accordingly. The view function reads and displays all event details from the file. Error handling is implemented for file operations and user inputs to ensure robust performance.

PSEUDOCODE

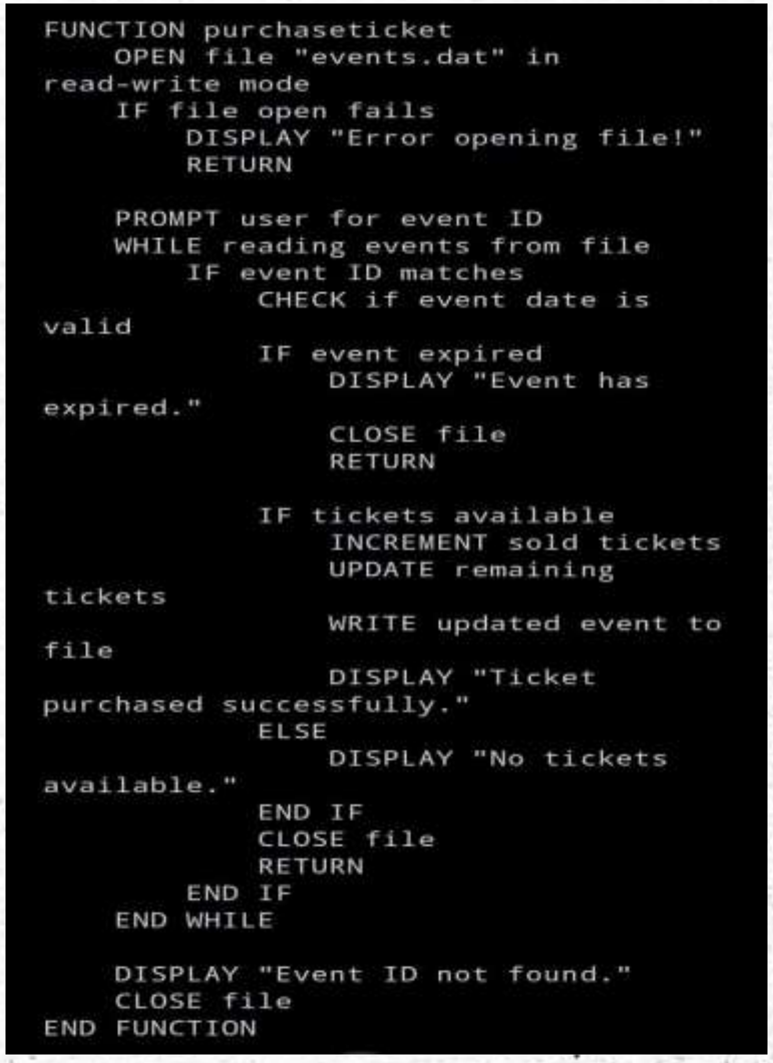


```
BEGIN

FUNCTION main
    WHILE TRUE
        CALL menu
    END WHILE
END FUNCTION

FUNCTION menu
    DISPLAY menu options
    READ user choice
    SWITCH choice
        CASE 1
            CALL addevent
        CASE 2
            CALL purchaseticket
        CASE 3
            CALL viewevents
        CASE 4
            DISPLAY "Exiting..."
            EXIT program
        DEFAULT
            DISPLAY "Invalid choice!
Please try again."
    END SWITCH
END FUNCTION
```

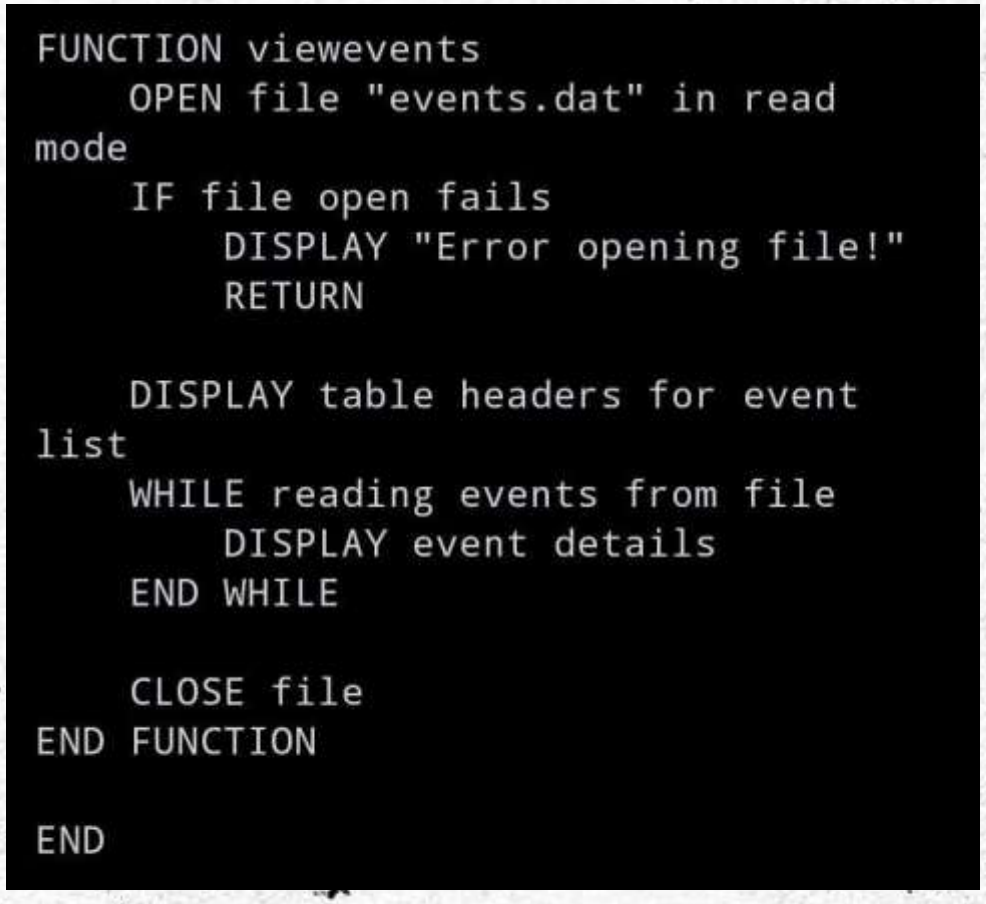


```
FUNCTION addevent
    OPEN file "events.dat" in append
mode
    IF file open fails
        DISPLAY "Error opening file!"
        RETURN

    PROMPT user for event details (ID,
name, total tickets, date)
    CHECK for duplicate ID
    IF duplicate ID found
        DISPLAY "Event ID already
exists."
        CLOSE file
        RETURN

    WRITE event details to file
    CLOSE file
    DISPLAY "Event added
successfully."
END FUNCTION
```

PSEUDOCODE



```
FUNCTION purchaseticket
    OPEN file "events.dat" in read-write mode
    IF file open fails
        DISPLAY "Error opening file!"
        RETURN

    PROMPT user for event ID
    WHILE reading events from file
        IF event ID matches
            CHECK if event date is valid
            IF event expired
                DISPLAY "Event has expired."
                CLOSE file
                RETURN

            IF tickets available
                INCREMENT sold tickets
                UPDATE remaining tickets
                WRITE updated event to file
                DISPLAY "Ticket purchased successfully."
            ELSE
                DISPLAY "No tickets available."
            END IF
            CLOSE file
            RETURN
        END IF
    END WHILE

    DISPLAY "Event ID not found."
    CLOSE file
END FUNCTION
```



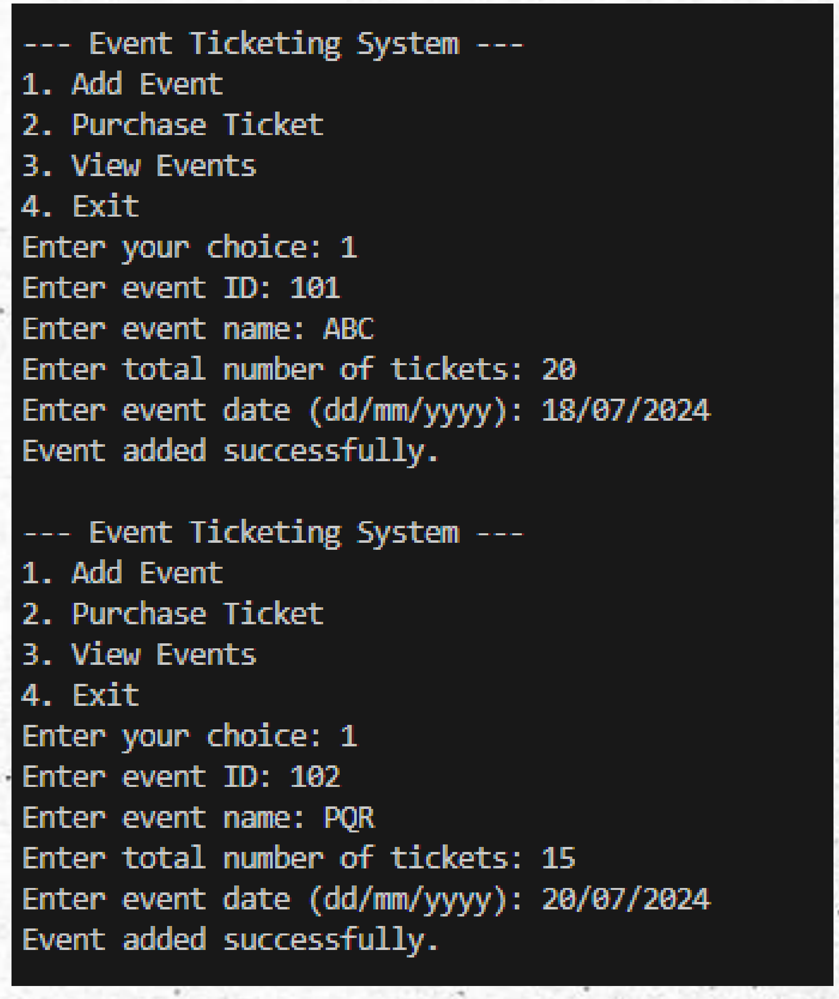
```
FUNCTION viewevents
    OPEN file "events.dat" in read mode
    IF file open fails
        DISPLAY "Error opening file!"
        RETURN

    DISPLAY table headers for event list
    WHILE reading events from file
        DISPLAY event details
    END WHILE

    CLOSE file
END FUNCTION

END
```

TESTING AND RESULTS

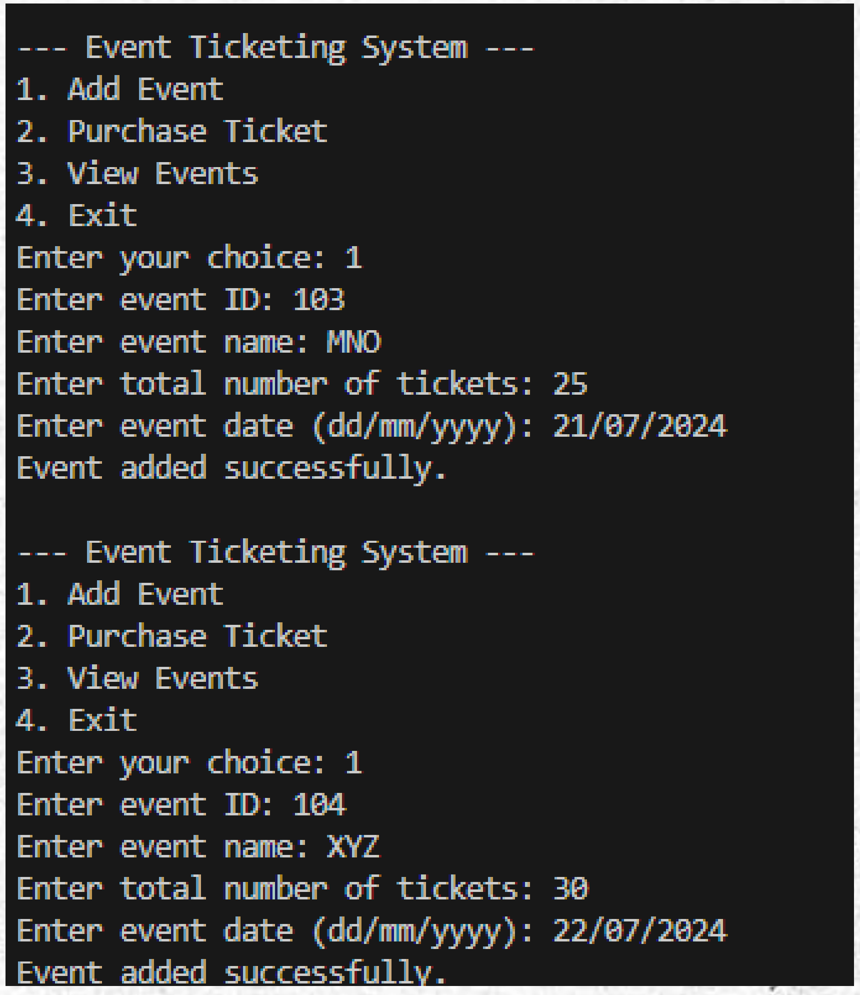
TEST CASES

1. **Add Event:** Verify that an event is added correctly by checking for unique IDs, proper event details input, and successful file writing. Test cases should include adding events with valid and invalid data (e.g., duplicate IDs).
2. **Purchase Ticket:** Validate that tickets can be purchased only if they are available, and ensure the system correctly updates sold and remaining tickets. Test cases should cover scenarios such as purchasing tickets for valid and expired events, as well as events with no available tickets.
3. **View Events:** Confirm that all event details are accurately displayed from the file. Test cases should include viewing events with various data to ensure correct formatting and data retrieval.
4. **Error Handling:** Test how the system handles file errors, invalid inputs, and edge cases such as incorrect dates or file read/write issues.

OUTPUT/RESULT

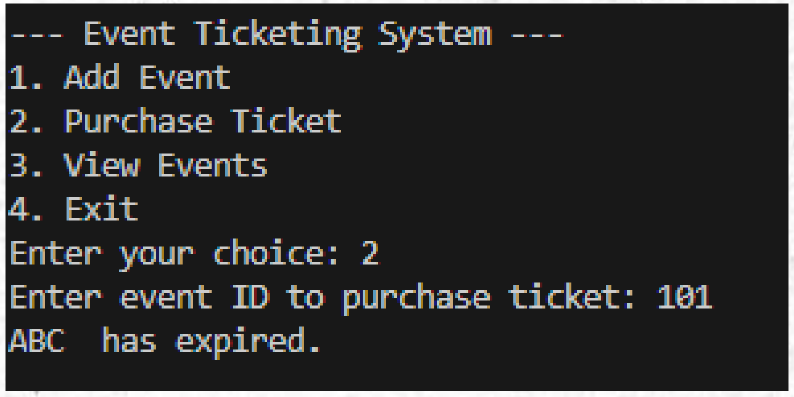


```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 1
Enter event ID: 101
Enter event name: ABC
Enter total number of tickets: 20
Enter event date (dd/mm/yyyy): 18/07/2024
Event added successfully.

--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 1
Enter event ID: 102
Enter event name: PQR
Enter total number of tickets: 15
Enter event date (dd/mm/yyyy): 20/07/2024
Event added successfully.
```

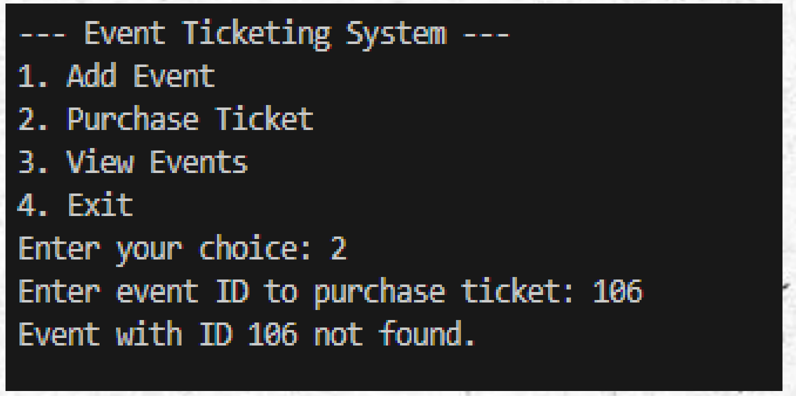


```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 1
Enter event ID: 103
Enter event name: MNO
Enter total number of tickets: 25
Enter event date (dd/mm/yyyy): 21/07/2024
Event added successfully.

--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 1
Enter event ID: 104
Enter event name: XYZ
Enter total number of tickets: 30
Enter event date (dd/mm/yyyy): 22/07/2024
Event added successfully.
```

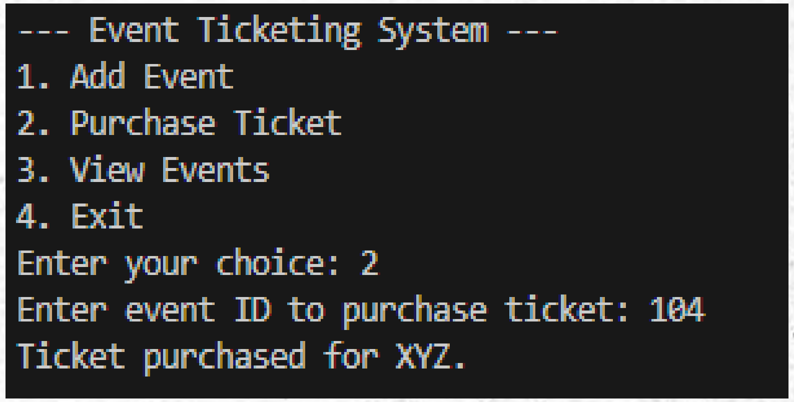
OUTPUT/RESULT



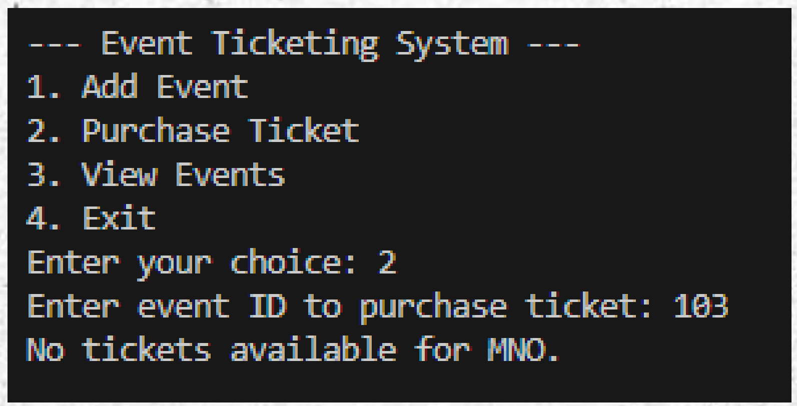
```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 2
Enter event ID to purchase ticket: 101
ABC  has expired.
```



```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 2
Enter event ID to purchase ticket: 106
Event with ID 106 not found.
```

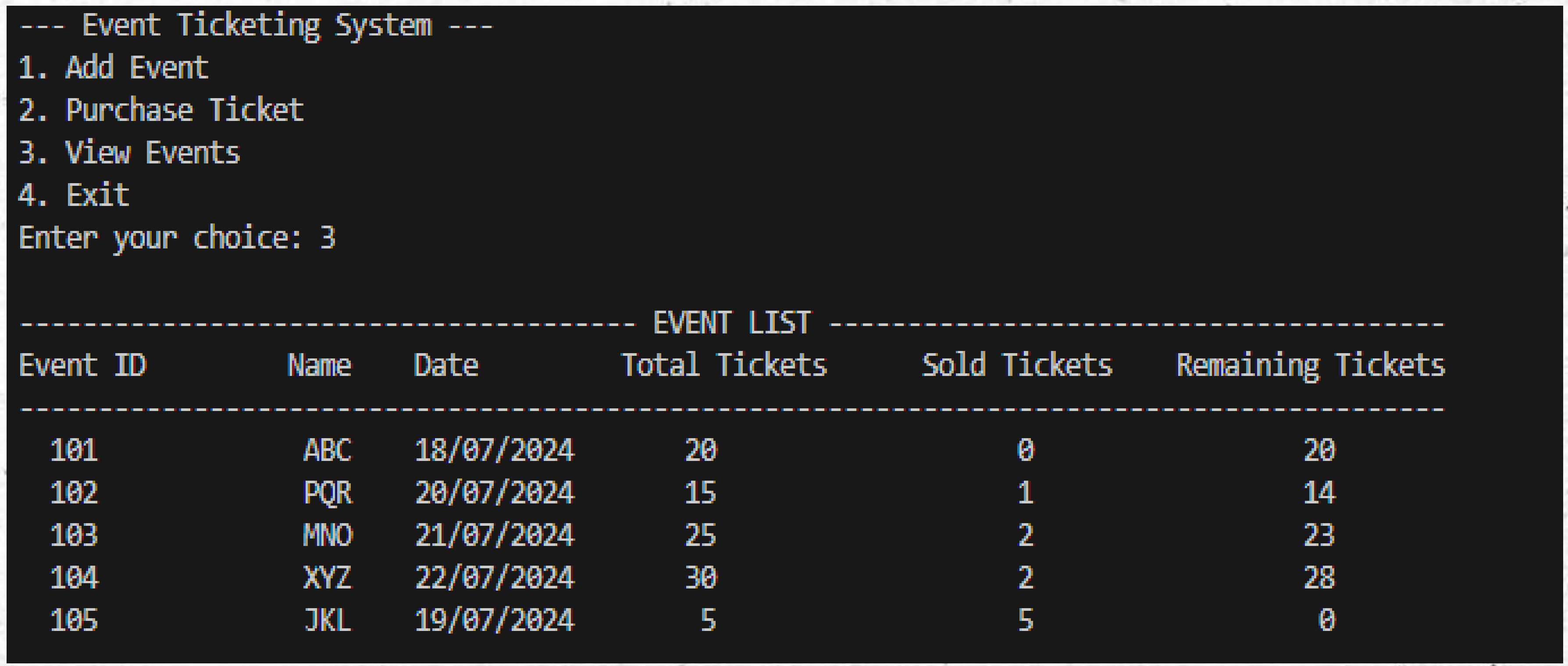


```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 2
Enter event ID to purchase ticket: 104
Ticket purchased for XYZ.
```



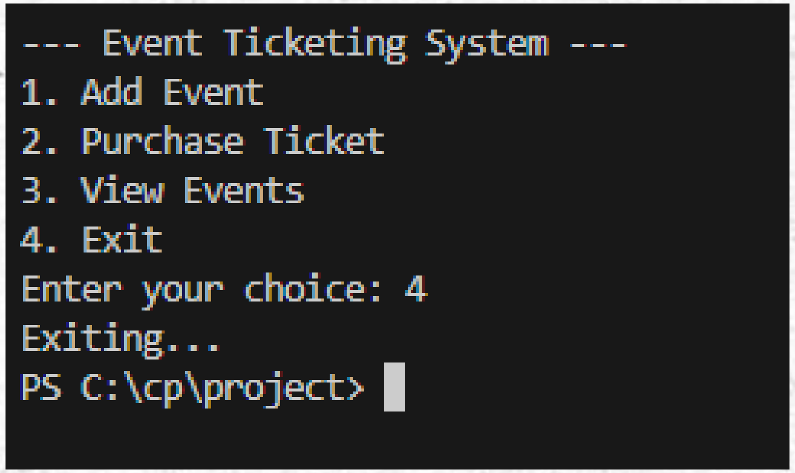
```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 2
Enter event ID to purchase ticket: 103
No tickets available for MNO.
```

OUTPUT/RESULT



```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 3

---------------------------------------- EVENT LIST ----------------------------------------
Event ID         Name     Date          Total Tickets      Sold Tickets    Remaining Tickets
---------------------------------------------------------------------------------------------
  101             ABC     18/07/2024       20                    0                  20
  102             PQR     20/07/2024       15                    1                  14
  103             MNO     21/07/2024       25                    2                  23
  104             XYZ     22/07/2024       30                    2                  28
  105             JKL     19/07/2024        5                    5                   0
```

OUTPUT/RESULT



```
--- Event Ticketing System ---
1. Add Event
2. Purchase Ticket
3. View Events
4. Exit
Enter your choice: 4
Exiting...
PS C:\cp\project>
```

RESULT DISCUSSION

The results of testing the Event Ticketing System confirm its effectiveness in managing event data and ticket sales. The system successfully handles various scenarios, including adding new events with unique IDs, purchasing tickets only when available, and displaying accurate event details. Error handling is robust, addressing issues such as duplicate IDs, expired events, and file access problems. The program's reliability in processing and updating event information demonstrates its practicality for automating ticket management, although further testing may be required to handle all edge cases and ensure optimal performance under different conditions.

CONCLUSION

The Event Ticketing System effectively automates the management of event details and ticket sales, offering a reliable solution for handling such tasks with improved accuracy and efficiency. By leveraging file storage for data persistence and implementing robust error handling, the system ensures smooth operation and minimizes manual errors. The successful implementation of core features—event addition, ticket purchasing, and event viewing—demonstrates the system's practicality for real-world use. Overall, this project highlights the benefits of automating event management processes and provides a foundation for further enhancements and scalability.

FUTURE ENHANCEMENTS

1. **User Authentication**: Add user accounts and role-based access.
2. **Web Interface**: Create a web-based user interface.
3. Database Integration: Use a relational database for data management.
4. **Real-Time Availability**: Update ticket availability in real-time.
5. **Payment Processing**: Integrate online payment options.
6. **Advanced Reporting**: Generate detailed sales and event reports.
7. **Notification System**: Implement alerts for ticket purchases and event updates.

THANK YOU...