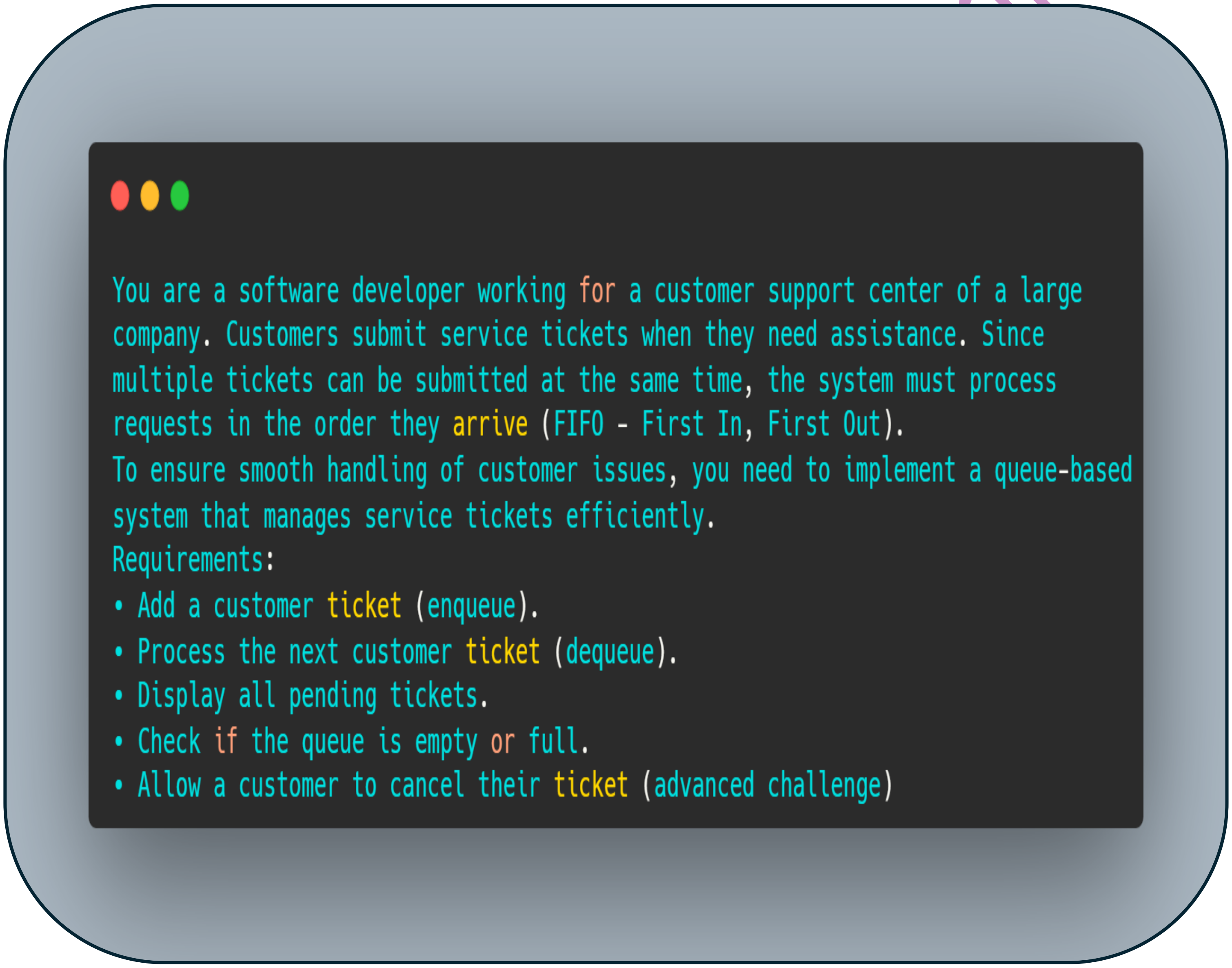
Lab Task 9 [Customer Service Ticketing System Using Queue]

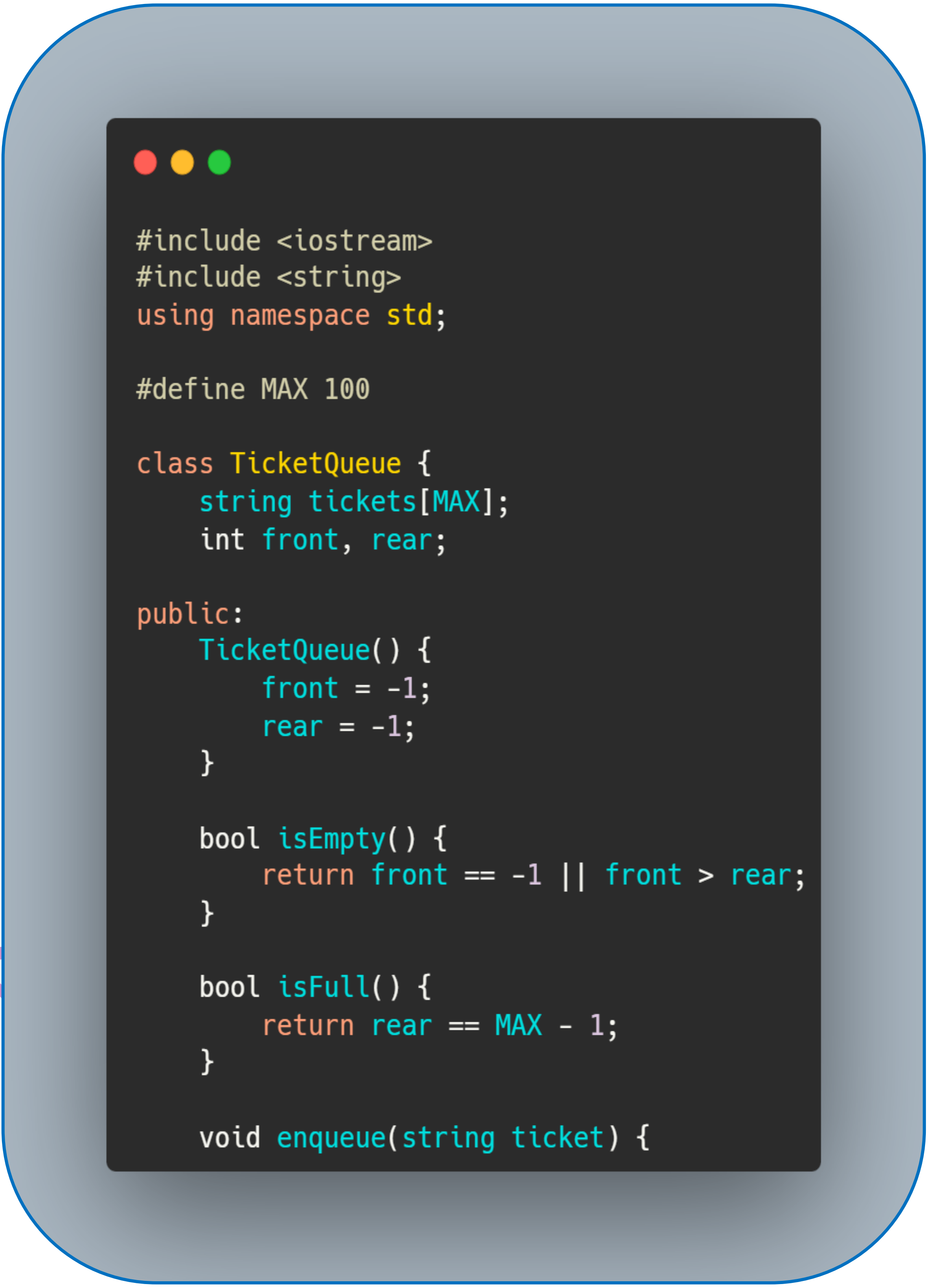
Task:



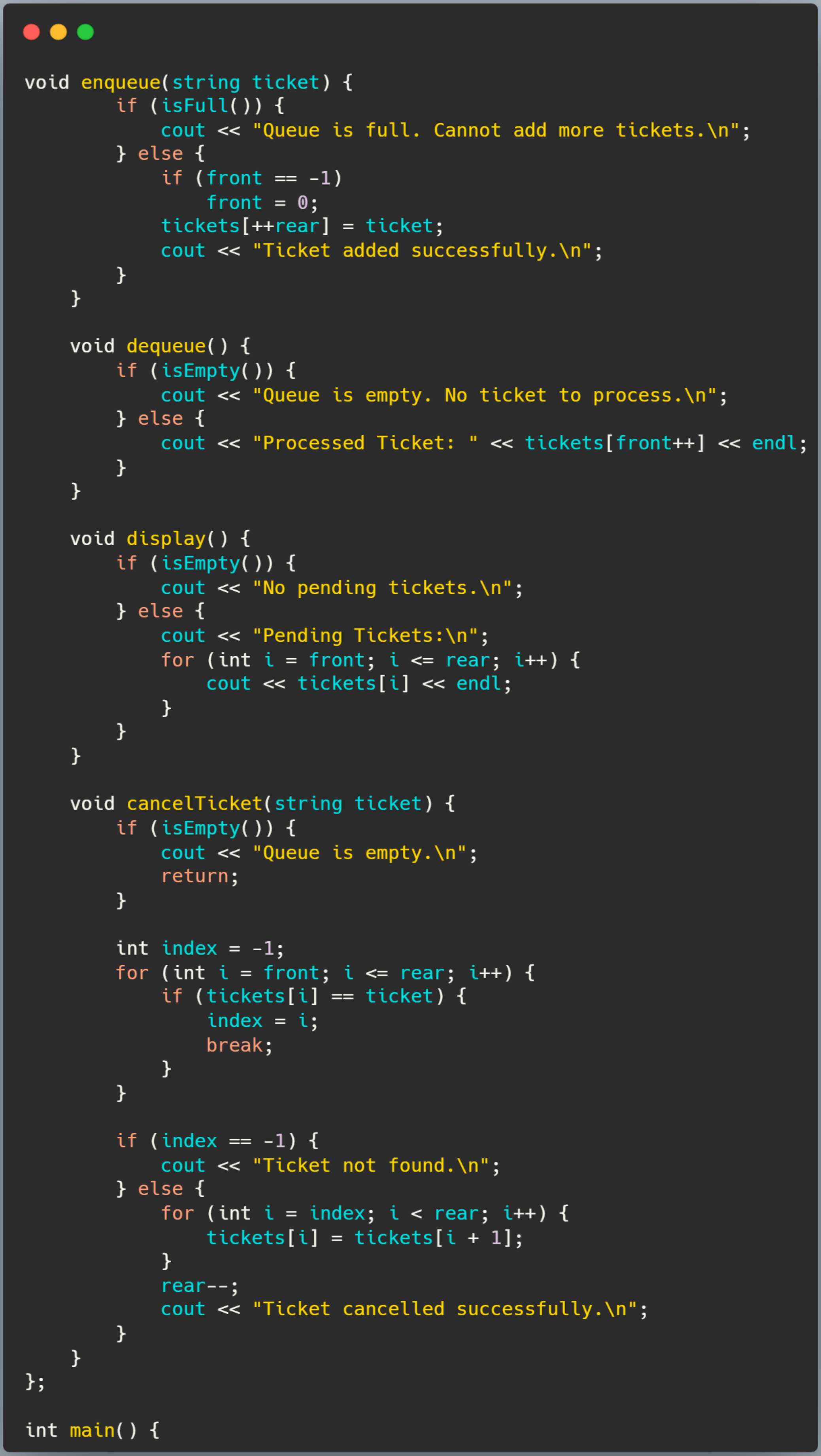
You are a software developer working for a customer support center of a large company. Customers submit service tickets when they need assistance. Since multiple tickets can be submitted at the same time, the system must process requests in the order they arrive (FIFO - First In, First Out).
To ensure smooth handling of customer issues, you need to implement a queue-based system that manages service tickets efficiently.
Requirements:

- Add a customer ticket (enqueue).
- Process the next customer ticket (dequeue).
- Display all pending tickets.
- Check if the queue is empty or full.
- Allow a customer to cancel their ticket (advanced challenge)

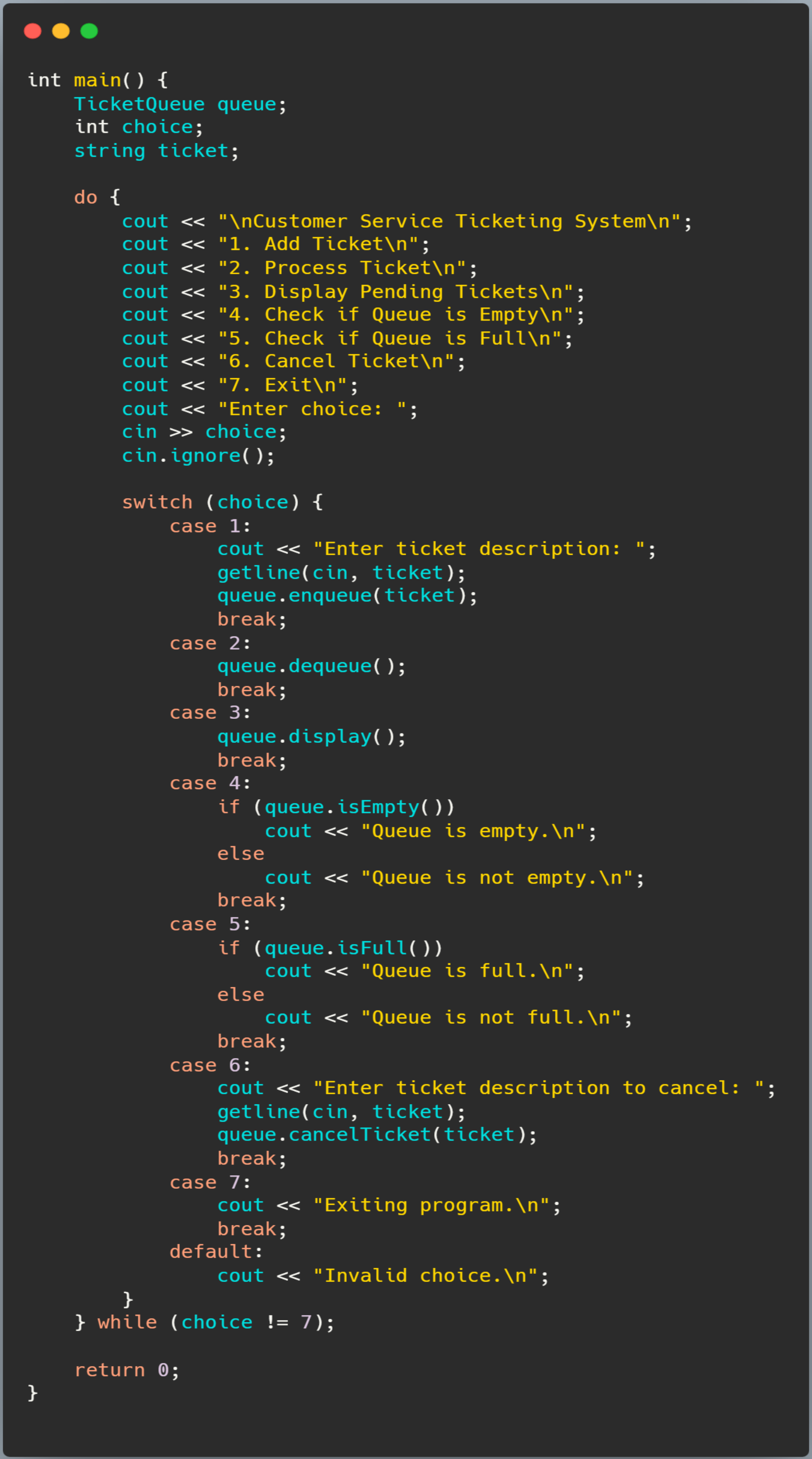
Code:



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

#define MAX 100

class TicketQueue {
    string tickets[MAX];
    int front, rear;

public:
    TicketQueue() {
        front = -1;
        rear = -1;
    }

    bool isEmpty() {
        return front == -1 || front > rear;
    }

    bool isFull() {
        return rear == MAX - 1;
    }

    void enqueue(string ticket) {
```



```cpp
void enqueue(string ticket) {
        if (isFull()) {
            cout << "Queue is full. Cannot add more tickets.\n";
        } else {
            if (front == -1)
                front = 0;
            tickets[++rear] = ticket;
            cout << "Ticket added successfully.\n";
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty. No ticket to process.\n";
        } else {
            cout << "Processed Ticket: " << tickets[front++] << endl;
        }
    }

    void display() {
        if (isEmpty()) {
            cout << "No pending tickets.\n";
        } else {
            cout << "Pending Tickets:\n";
            for (int i = front; i <= rear; i++) {
                cout << tickets[i] << endl;
            }
        }
    }

    void cancelTicket(string ticket) {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return;
        }

        int index = -1;
        for (int i = front; i <= rear; i++) {
            if (tickets[i] == ticket) {
                index = i;
                break;
            }
        }

        if (index == -1) {
            cout << "Ticket not found.\n";
        } else {
            for (int i = index; i < rear; i++) {
                tickets[i] = tickets[i + 1];
            }
            rear--;
            cout << "Ticket cancelled successfully.\n";
        }
    }
};

int main() {
```

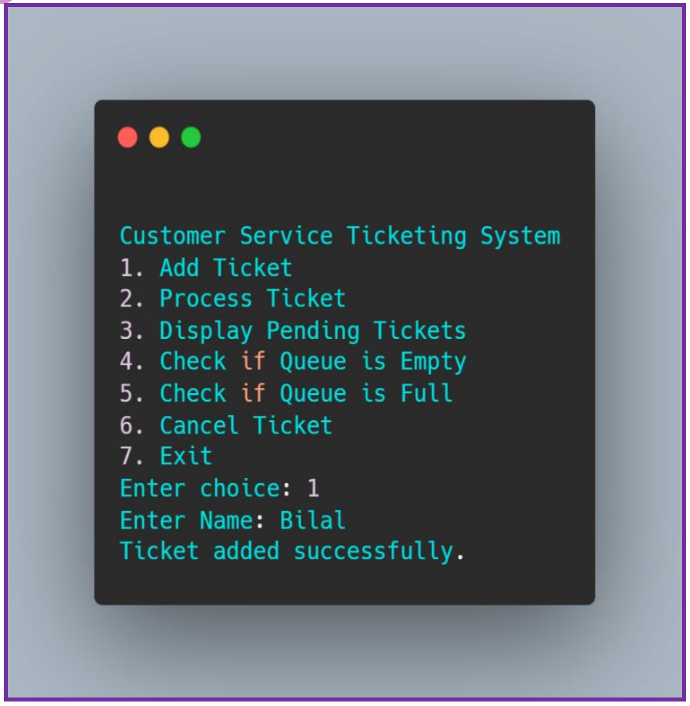


```cpp
int main() {
    TicketQueue queue;
    int choice;
    string ticket;

    do {
        cout << "\nCustomer Service Ticketing System\n";
        cout << "1. Add Ticket\n";
        cout << "2. Process Ticket\n";
        cout << "3. Display Pending Tickets\n";
        cout << "4. Check if Queue is Empty\n";
        cout << "5. Check if Queue is Full\n";
        cout << "6. Cancel Ticket\n";
        cout << "7. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;
        cin.ignore();

        switch (choice) {
            case 1:
                cout << "Enter ticket description: ";
                getline(cin, ticket);
                queue.enqueue(ticket);
                break;
            case 2:
                queue.dequeue();
                break;
            case 3:
                queue.display();
                break;
            case 4:
                if (queue.isEmpty())
                    cout << "Queue is empty.\n";
                else
                    cout << "Queue is not empty.\n";
                break;
            case 5:
                if (queue.isFull())
                    cout << "Queue is full.\n";
                else
                    cout << "Queue is not full.\n";
                break;
            case 6:
                cout << "Enter ticket description to cancel: ";
                getline(cin, ticket);
                queue.cancelTicket(ticket);
                break;
            case 7:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice.\n";
        }
    } while (choice != 7);

    return 0;
}
```

Table Of Output:

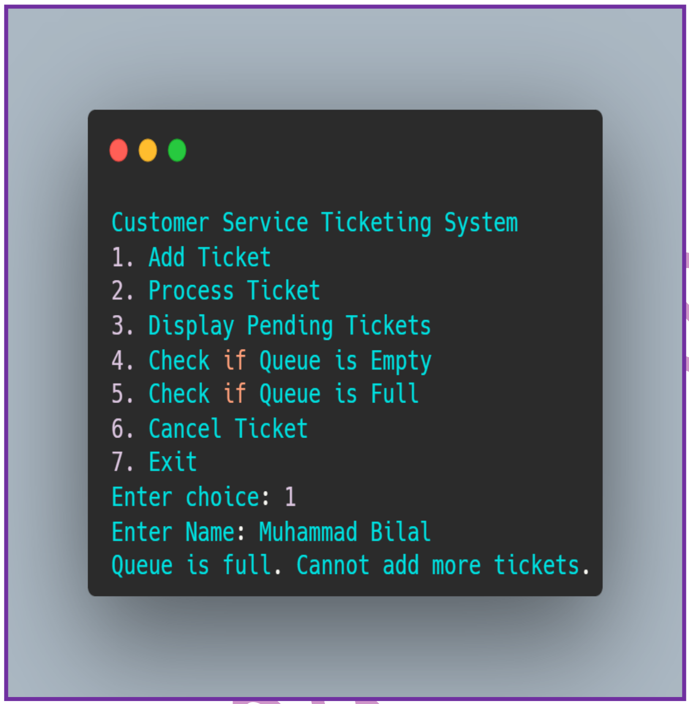
	Condition	Output
1	Enqueue ticket when queue is **not full**	Ticket added successfully.
2	Enqueue ticket when queue is **full**	Queue is full. Cannot add more tickets.
3	Dequeue ticket when queue is **not empty**	Processed Ticket: <ticket description>
4	Dequeue ticket when queue is **empty**	Queue is empty. No ticket to process.
5	Display tickets when queue is **not empty**	Pending Tickets printed
6	Display tickets when queue is **empty**	No pending tickets.
7	Cancel ticket when queue is **empty**	Queue is empty.
8	Cancel ticket when ticket **exists in queue**	Ticket cancelled successfully.
9	Cancel ticket when ticket **does not exist**	Ticket not found.
10	Check empty when queue **is empty**	Queue is empty.
11	Check empty when queue **is not empty**	Queue is not empty.
12	Check full when queue **is full**	Queue is full.
13	Check full when queue **is not full**	Queue is not full.
14	User enters **Exit (7)**	Exiting program.
15	User enters **invalid menu choice**	Invalid choice.
	Total 15 Possible Outputs	

Enqueue ticket when queue is **not full**



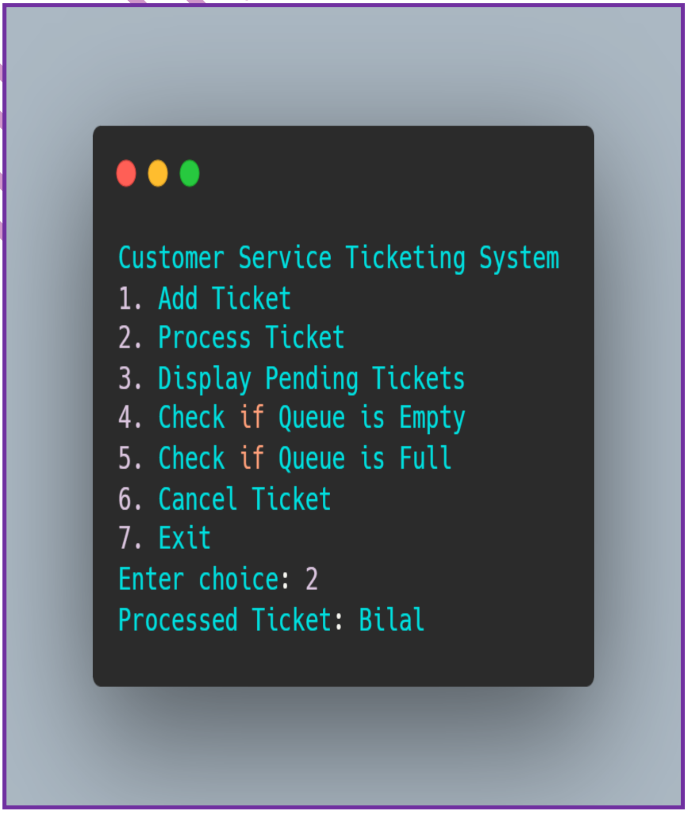
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 1
Enter Name: Bilal
Ticket added successfully.
```

Enqueue ticket when queue is **full**



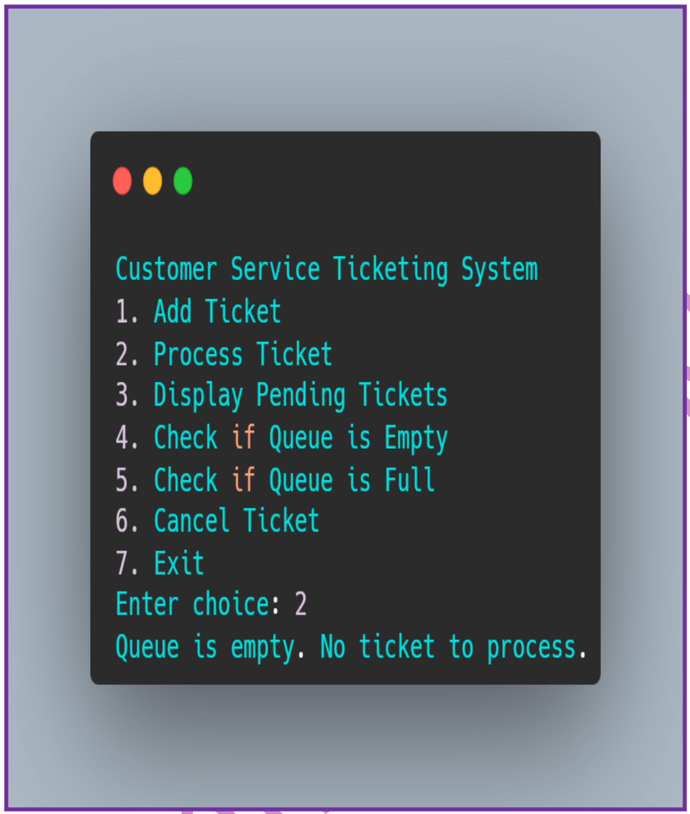
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 1
Enter Name: Muhammad Bilal
Queue is full. Cannot add more tickets.
```

Dequeue ticket when queue is **not empty**



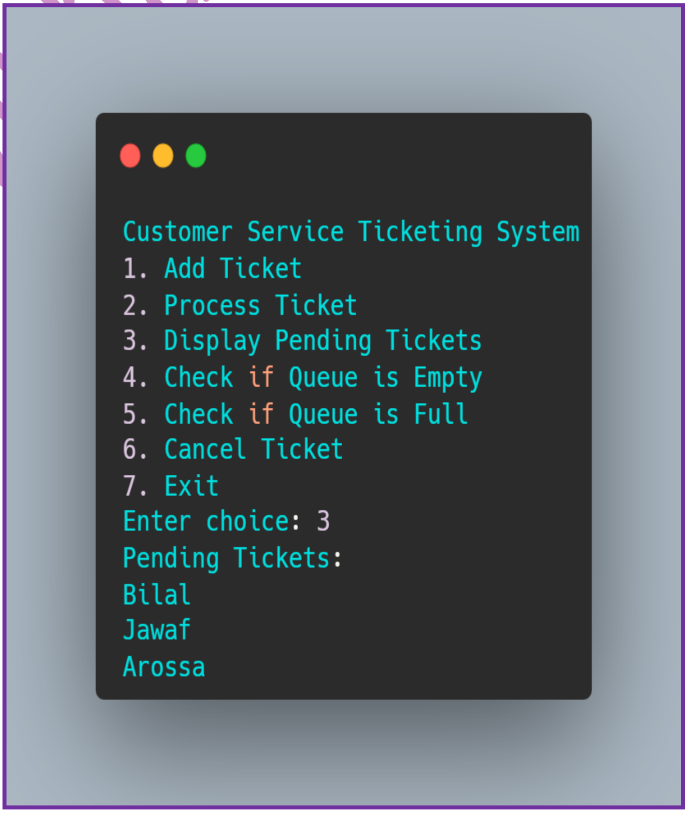
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 2
Processed Ticket: Bilal
```

Dequeue ticket when queue is **empty**



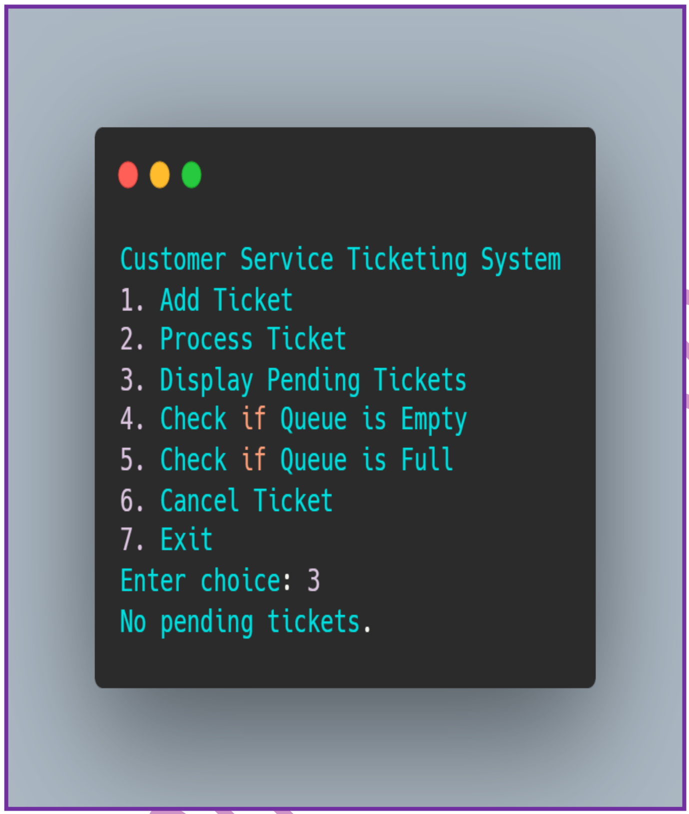
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 2
Queue is empty. No ticket to process.
```

Display tickets when queue is **not empty**



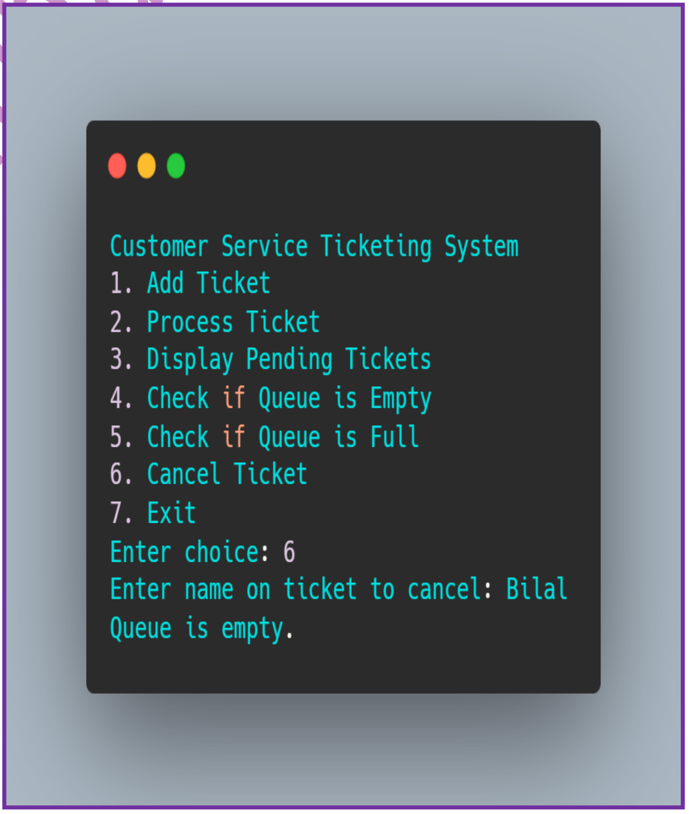
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 3
Pending Tickets:
Bilal
Jawaf
Arossa
```

Display tickets when queue is **empty**



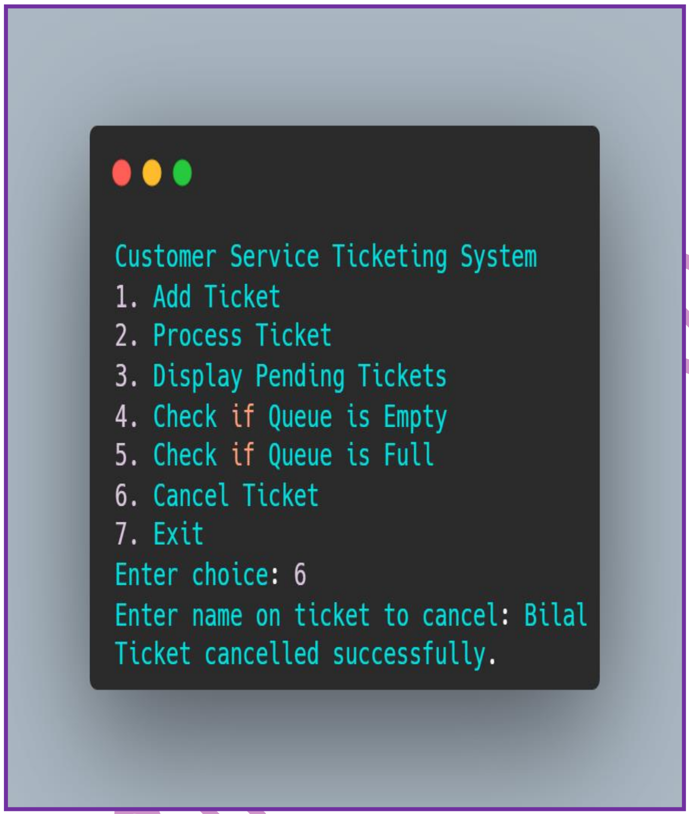
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 3
No pending tickets.
```

Cancel ticket when queue is **empty**



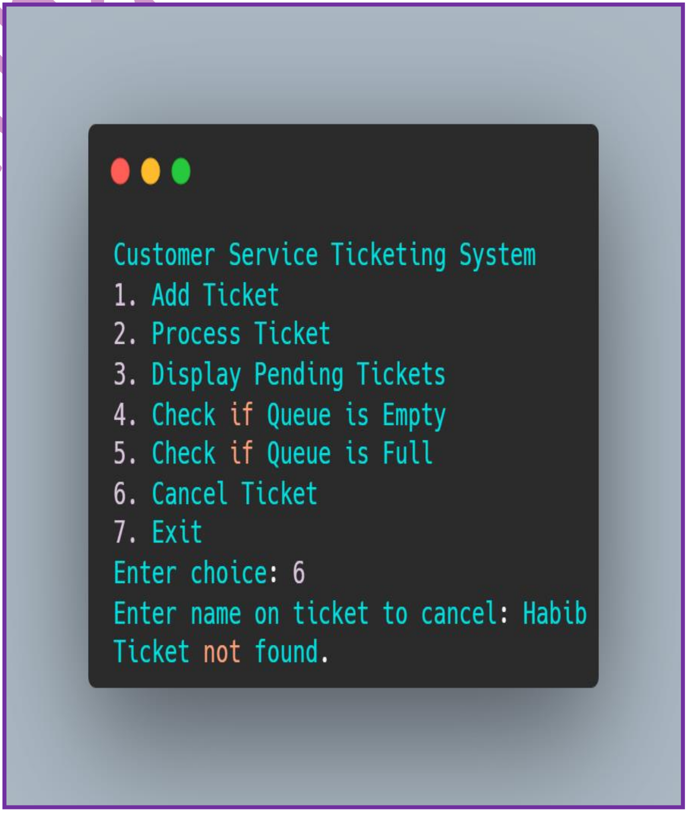
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 6
Enter name on ticket to cancel: Bilal
Queue is empty.
```

Cancel ticket when ticket **exists in queue**



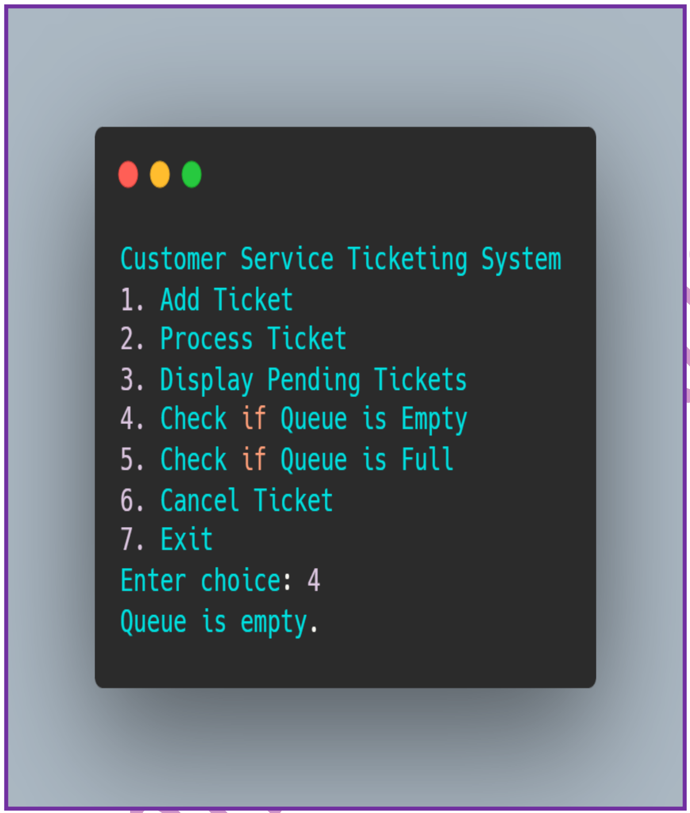
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 6
Enter name on ticket to cancel: Bilal
Ticket cancelled successfully.
```

Cancel ticket when ticket **does not exist**



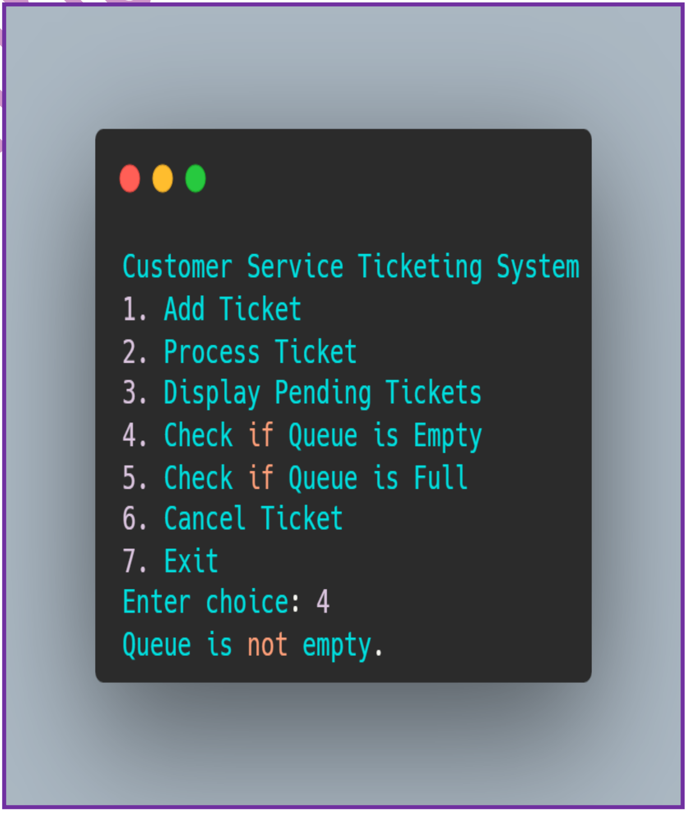
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 6
Enter name on ticket to cancel: Habib
Ticket not found.
```

Check empty when queue **is empty**



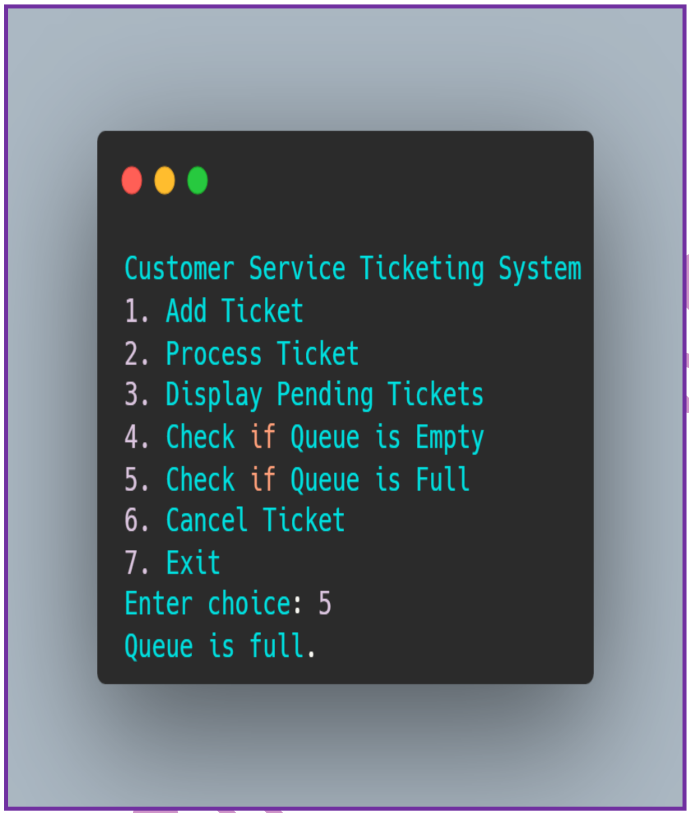
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 4
Queue is empty.
```

Check empty when queue **is not empty**



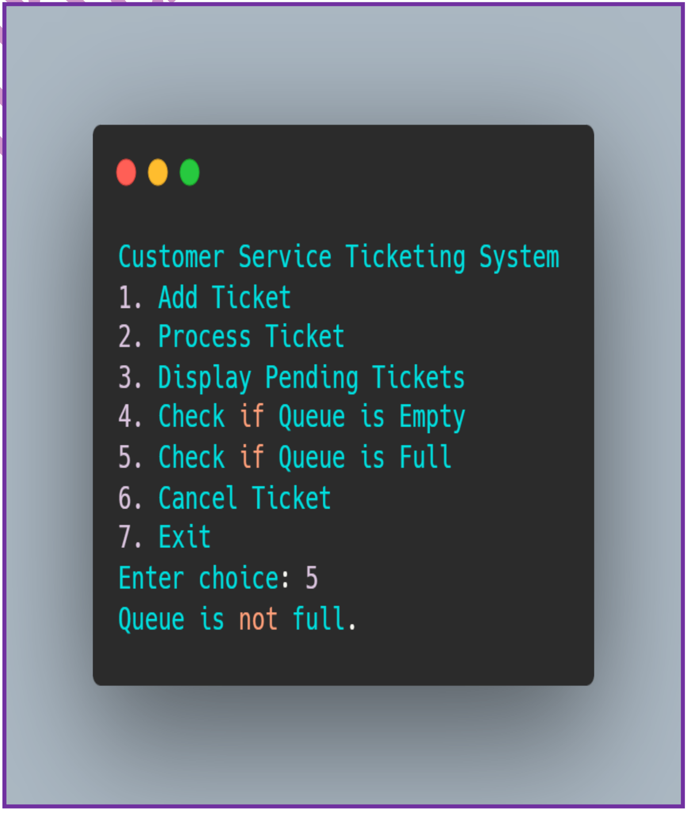
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 4
Queue is not empty.
```

Check full **when** queue **is full**



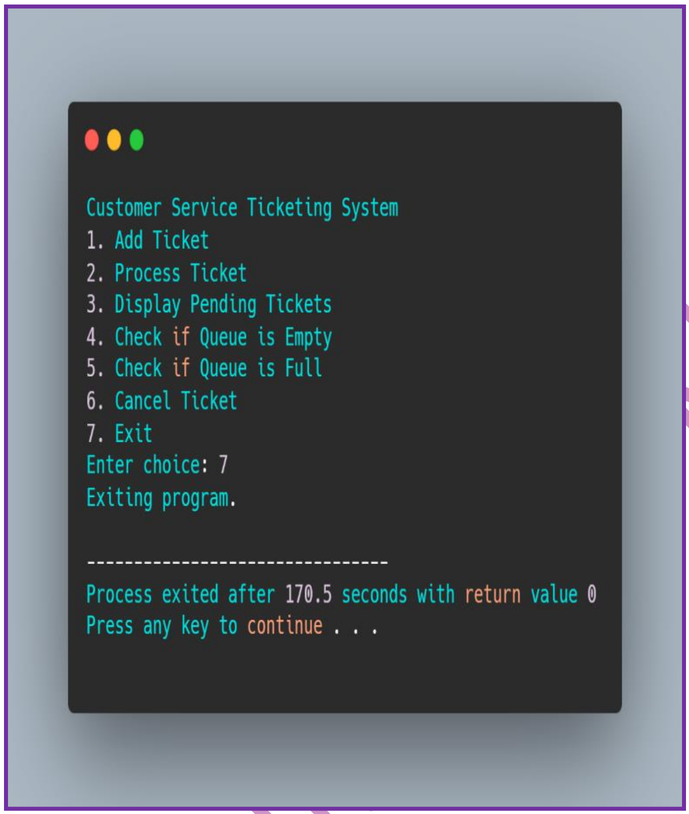
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 5
Queue is full.
```

Check full **when** queue **is not full**



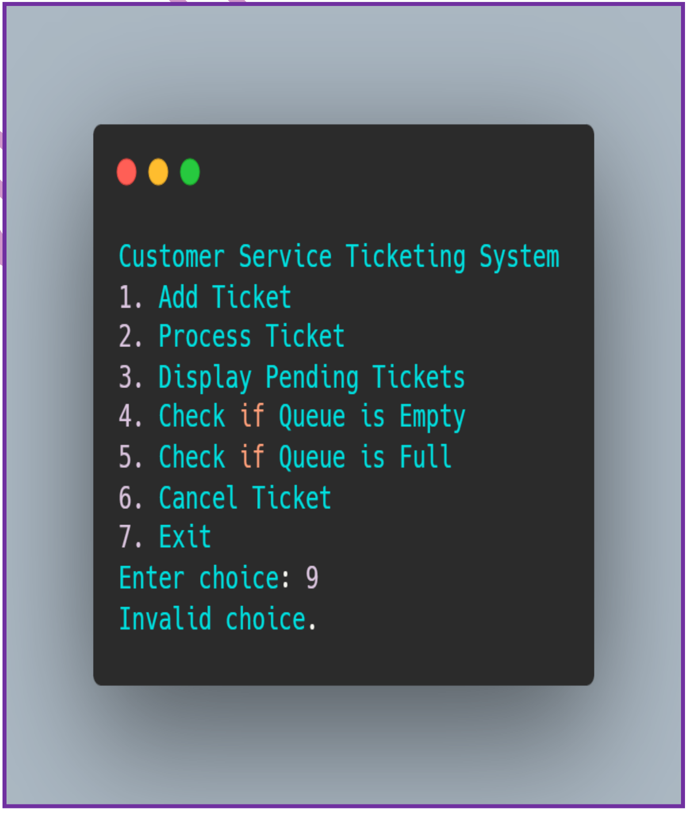
```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 5
Queue is not full.
```

When enter **Exit (7)**



```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 7
Exiting program.

--------------------------------
Process exited after 170.5 seconds with return value 0
Press any key to continue . . .
```

When invalid **menu choice**



```
Customer Service Ticketing System
1. Add Ticket
2. Process Ticket
3. Display Pending Tickets
4. Check if Queue is Empty
5. Check if Queue is Full
6. Cancel Ticket
7. Exit
Enter choice: 9
Invalid choice.
```

Click here to Get this code on GitHub

Click here to Test this Code by Yourself.

Conclusion: -

The Customer Service Ticketing System implemented using an array-based queue can generate **15 distinct logical outputs**. These outputs reflect correct handling of queue operations such as insertion, deletion, searching, and state checking, fulfilling standard.