Review the key concepts covered during the activity.

Key Points from the Activity

Stacks and Queues

- **Stack:** Adds and removes items from the top. Like a stack of books.
- **Queue:** Adds items at the back and removes from the front. Like a line at a store.

Making a Queue with Two Stacks

- **Enqueue:** Add customer to stackNewestOnTop.
- **Dequeue:** Move items to stackOldestOnTop when needed, then remove from the top of stackOldestOnTop.
- **Peek:** Look at the next customer without removing them.

Efficiency

Using two stacks makes sure adding and serving customers is fast overall.

Object-Oriented Programming (OOP)

- **Class Design:** TicketCounter class handles everything related to the queue.
- **Encapsulation:** Keeps the stacks hidden and uses methods to interact with them.

Error Handling

Methods check if the queue is empty and handle it nicely.

User Interaction

- **Menu Interface:** Simple text menu lets users choose what to do.
- **Input Handling:** Users can add, serve, peek, and see the queue status through menu options.

Summary

I built a ticket system using two stacks to act like a queue. It covers basic concepts of stacks, queues, and OOP, and includes a simple menu for user interaction.

Discuss the importance of queues in managing processes and scenarios like ticket counters.

Importance of Queues

Managing Processes

- **Order:** Queues keep things in order. First come, first served. Just like waiting in line at a fast food place.
- **Fairness:** Everyone gets their turn based on when they arrived. No cutting in line!

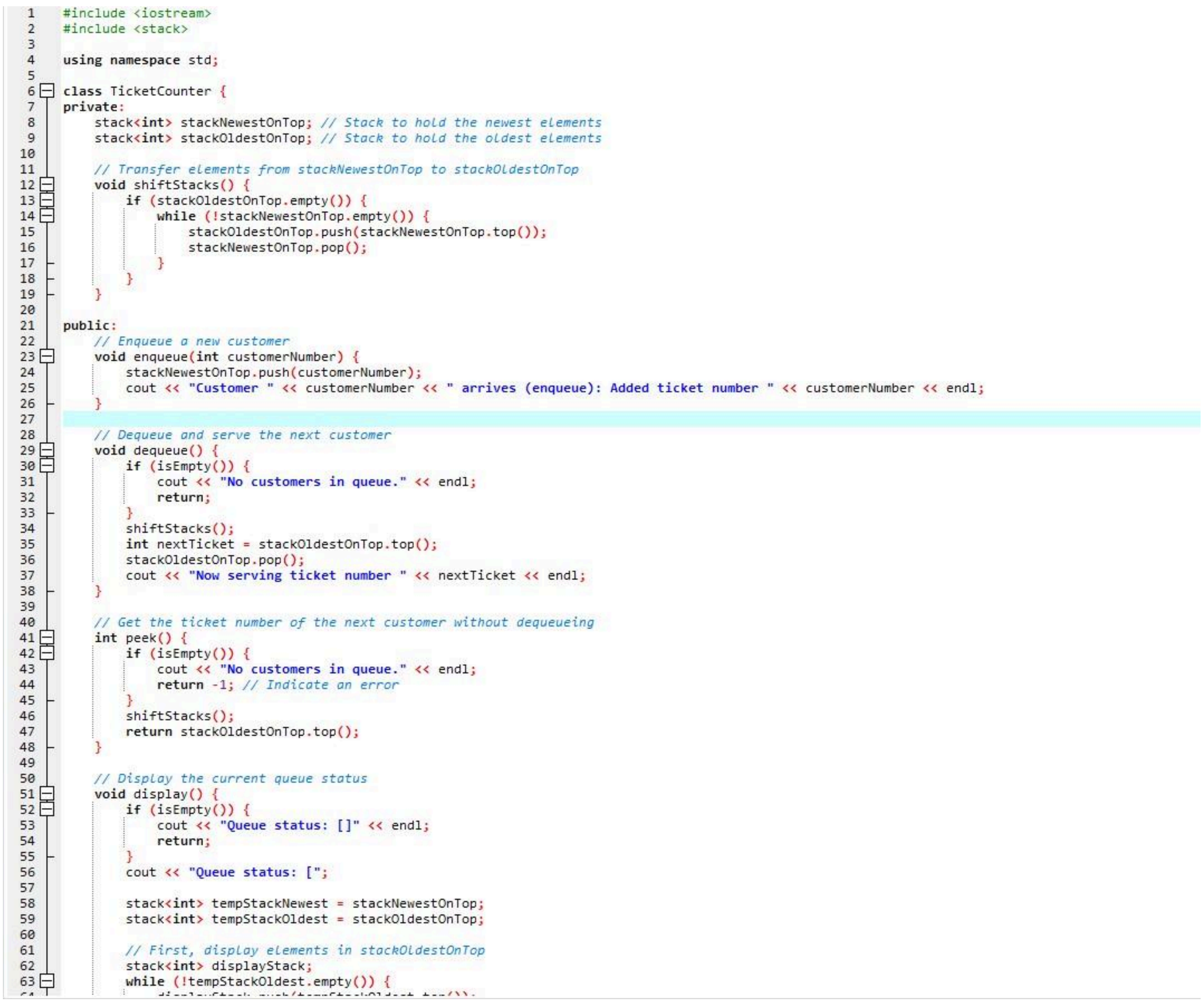
Scenarios like Ticket Counters

- **Efficiency:** Helps manage crowds efficiently. Each person is served one by one without confusion.
- **Simplicity**: Easy to understand and follow. Customers know their spot in line.
- **Predictability:** You can estimate waiting time since it's clear who's next.

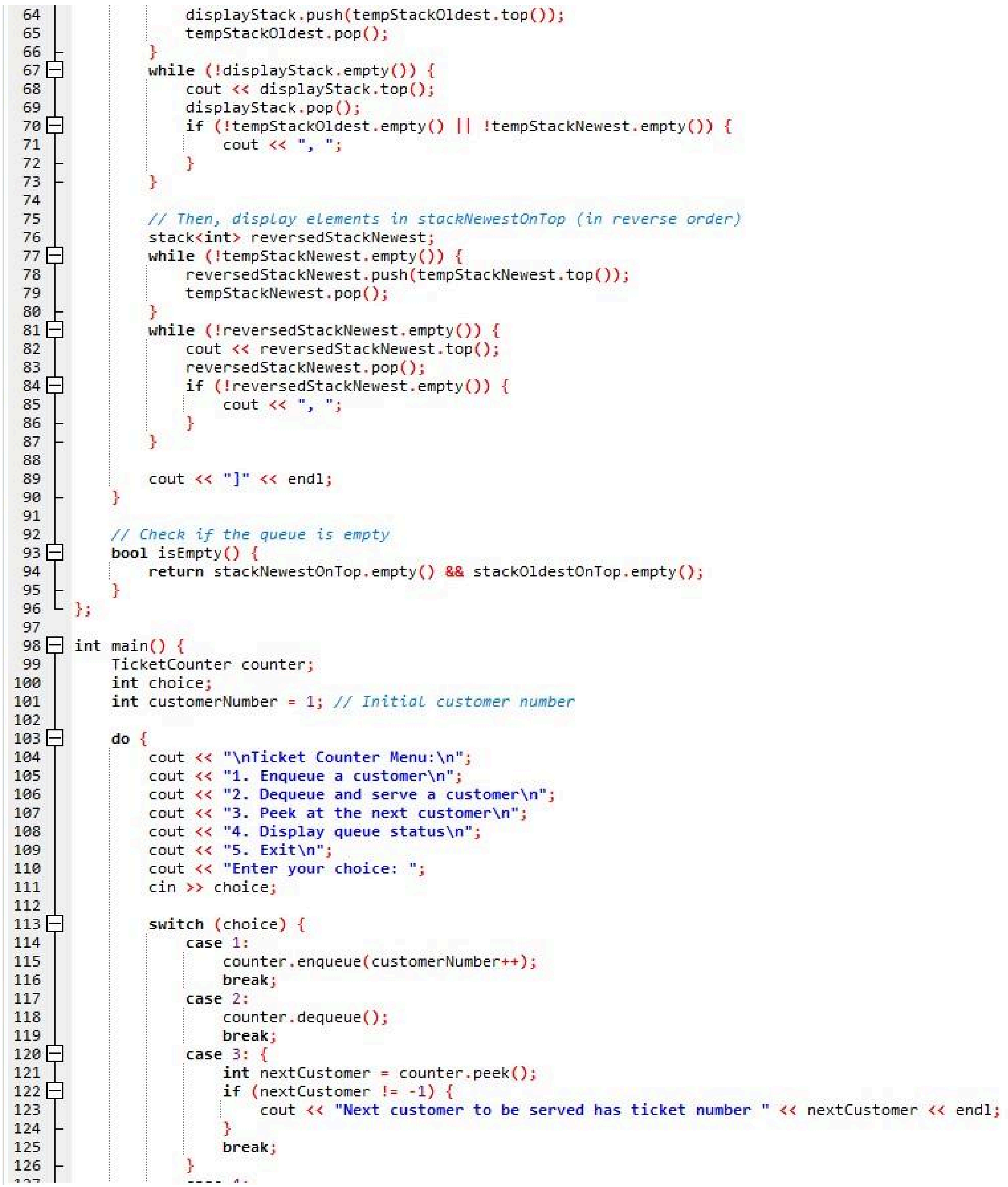
Summary

Queues are essential for keeping processes smooth and fair, especially in places like ticket counters. They ensure everyone gets served in the order they arrive, making the system efficient and easy to manage.

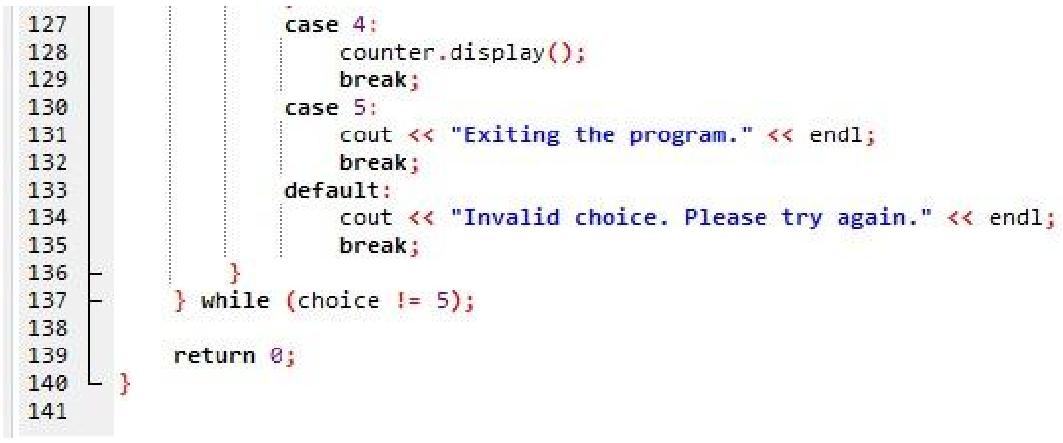
https://github.com/jims-sama/ticketingsystem



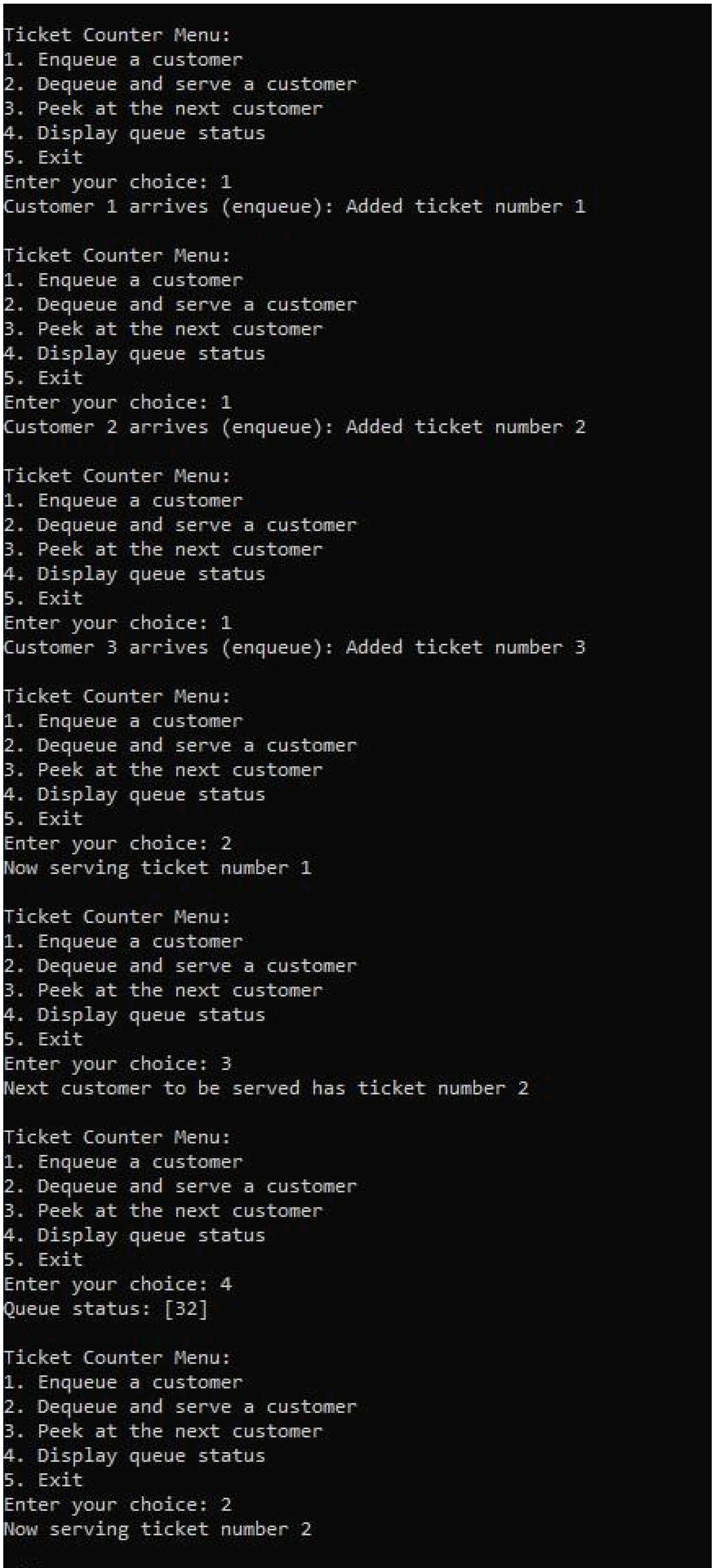
```cpp
#include <iostream>
#include <stack>

using namespace std;

class TicketCounter {
private:
    stack<int> stackNewestOnTop; // Stack to hold the newest elements
    stack<int> stackOldestOnTop; // Stack to hold the oldest elements

    // Transfer elements from stackNewestOnTop to stackOldestOnTop
    void shiftStacks() {
        if (stackOldestOnTop.empty()) {
            while (!stackNewestOnTop.empty()) {
                stackOldestOnTop.push(stackNewestOnTop.top());
                stackNewestOnTop.pop();
            }
        }
    }

public:
    // Enqueue a new customer
    void enqueue(int customerNumber) {
        stackNewestOnTop.push(customerNumber);
        cout << "Customer " << customerNumber << " arrives (enqueue): Added ticket number " << customerNumber << endl;
    }

    // Dequeue and serve the next customer
    void dequeue() {
        if (isEmpty()) {
            cout << "No customers in queue." << endl;
            return;
        }
        shiftStacks();
        int nextTicket = stackOldestOnTop.top();
        stackOldestOnTop.pop();
        cout << "Now serving ticket number " << nextTicket << endl;
    }

    // Get the ticket number of the next customer without dequeueing
    int peek() {
        if (isEmpty()) {
            cout << "No customers in queue." << endl;
            return -1; // Indicate an error
        }
        shiftStacks();
        return stackOldestOnTop.top();
    }

    // Display the current queue status
    void display() {
        if (isEmpty()) {
            cout << "Queue status: []" << endl;
            return;
        }
        cout << "Queue status: [";

        stack<int> tempStackNewest = stackNewestOnTop;
        stack<int> tempStackOldest = stackOldestOnTop;

        // First, display elements in stackOldestOnTop
        stack<int> displayStack;
        while (!tempStackOldest.empty()) {
            displayStack.push(tempStackOldest.top());
```



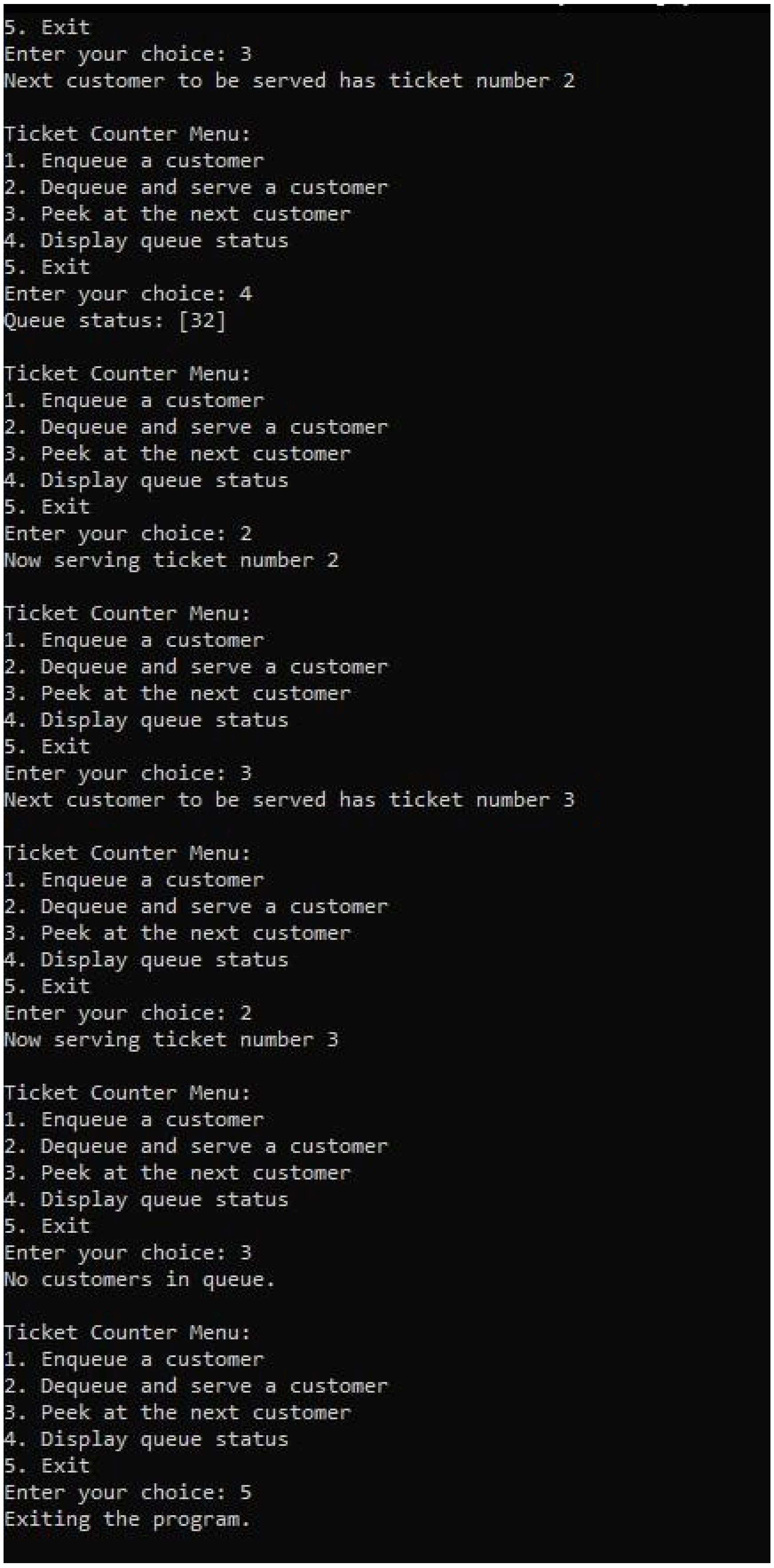
```cpp
            displayStack.push(tempStackOldest.top());
            tempStackOldest.pop();
        }
        while (!displayStack.empty()) {
            cout << displayStack.top();
            displayStack.pop();
            if (!tempStackOldest.empty() || !tempStackNewest.empty()) {
                cout << ", ";
            }
        }

        // Then, display elements in stackNewestOnTop (in reverse order)
        stack<int> reversedStackNewest;
        while (!tempStackNewest.empty()) {
            reversedStackNewest.push(tempStackNewest.top());
            tempStackNewest.pop();
        }
        while (!reversedStackNewest.empty()) {
            cout << reversedStackNewest.top();
            reversedStackNewest.pop();
            if (!reversedStackNewest.empty()) {
                cout << ", ";
            }
        }

        cout << "]" << endl;
    }

    // Check if the queue is empty
    bool isEmpty() {
        return stackNewestOnTop.empty() && stackOldestOnTop.empty();
    }
};

int main() {
    TicketCounter counter;
    int choice;
    int customerNumber = 1; // Initial customer number

    do {
        cout << "\nTicket Counter Menu:\n";
        cout << "1. Enqueue a customer\n";
        cout << "2. Dequeue and serve a customer\n";
        cout << "3. Peek at the next customer\n";
        cout << "4. Display queue status\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                counter.enqueue(customerNumber++);
                break;
            case 2:
                counter.dequeue();
                break;
            case 3: {
                int nextCustomer = counter.peek();
                if (nextCustomer != -1) {
                    cout << "Next customer to be served has ticket number " << nextCustomer << endl;
                }
                break;
            }
```



```cpp
            case 4:
                counter.display();
                break;
            case 5:
                cout << "Exiting the program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }
    } while (choice != 5);

    return 0;
}
```



```
Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 1
Customer 1 arrives (enqueue): Added ticket number 1

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 1
Customer 2 arrives (enqueue): Added ticket number 2

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 1
Customer 3 arrives (enqueue): Added ticket number 3

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 2
Now serving ticket number 1

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 3
Next customer to be served has ticket number 2

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 4
Queue status: [32]

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 2
Now serving ticket number 2
```


```
5. Exit
Enter your choice: 3
Next customer to be served has ticket number 2

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 4
Queue status: [32]

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 2
Now serving ticket number 2

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 3
Next customer to be served has ticket number 3

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 2
Now serving ticket number 3

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 3
No customers in queue.

Ticket Counter Menu:
1. Enqueue a customer
2. Dequeue and serve a customer
3. Peek at the next customer
4. Display queue status
5. Exit
Enter your choice: 5
Exiting the program.
```