Introduction:

Hello, in this video, we will be discussing the code for a customer queue application in C#. Specifically, we will compare two versions of this application - Task B and Task C. We will go over the differences between the two versions and explain why Task C is an improvement over Task B.

Task B:

Let's start with Task B. This version of the application has a queue data structure that holds a fixed number of customer objects. The queue can only hold a maximum number of customers, and if this limit is reached, no new customers can be added. In addition, there is no provision to remove any customers from the queue.

Task C:

In Task C, the customer queue has been improved by using a circular buffer to hold the data. This ensures that the tail position is reset to 0 when the queue becomes empty, so that new customers can be added to the queue even if it was previously full. In addition, a new method has been added to the queue data structure - ReverseKElements(int k) - which reverses the first k elements in the queue.

Explanation:

Let's take a closer look at Task C. First, we see that the Customer class now has an additional attribute - Age. This allows us to store additional information about each customer.

Next, we look at the QueueDataStructure class. The store attribute is now an array of Customer objects that can hold a maximum of five customers. However, this limit can be changed by updating the value of the maxSize attribute in the constructor.

The Enqueue() method now checks if the queue is full before adding a new customer. If the queue is full, a warning message is displayed, and the customer is not added to the queue. Otherwise, the new customer is added to the tail of the queue, and the tailIndex is updated accordingly.

The Dequeue() method removes the head element from the queue and updates the headIndex accordingly.

The Peek() method returns the head element from the queue without removing it.

The IsEmpty() and IsFull() methods check if the queue is empty or full, respectively.

Finally, the ReverseKElements() method reverses the first k elements in the queue. This method first checks if there are enough elements in the queue to reverse. If not, an error message is displayed, and the method returns. Otherwise, a temporary stack is created to hold the first k elements in the queue. These elements are then popped off the stack and added back to the queue in reverse order.

The Form1 class is responsible for handling the user interface. It has an instance of the QueueDataStructure class, which is initialized with a maximum size of five. The Enqueue\_Click() method checks if the age input is a valid integer before adding the new customer to the queue. The ReverseButton\_Click() method calls the ReverseKElements() method of the QueueDataStructure class to reverse the first k elements in the queue.

Conclusion:

In conclusion, Task C is an improvement over Task B because it allows for the addition and removal of customers to the queue even if it was previously full. In addition, it provides the ability to reverse the first k elements in the queue. This makes the application more flexible and allows for better management of the customer queue.