Class / Lab Assignment-2

Data Structures – Stacks and Queues (Static & Dynamic Implementations with Real-World Case Studies)

Course Learning Outcomes (CLOs) Mapping

• Class Part (Theory) → CLO-1: Employ linear data structures to solve computing problems.

• Lab Part (Implementation in C++) → CLO-4: Implement various data structures, searching, sorting, and hashing in a programming language.

Problem Statement

This assignment explores Stacks and Queues using both static (array-based) and dynamic (linked-list-based) representations. Each problem is built around a real-world scenario to demonstrate how linear data structures power everyday systems in e-commerce, healthcare, logistics, and technology.

Q#1: E-Commerce Return Management System (Static Stack Implementation)

Major online retailers such as Daraz and Amazon process hundreds of product return requests every hour. The Return Management Department handles these requests in reverse order of arrival to quickly resolve the most recent issues first.

Design and implement a static stack (array-based) system that stores customer return requests.
Each request includes:
• Order ID
• Customer Name
• Return Reason
• Date of Request

Operations:
1. pushReturnRequest() – Add a new return request to the stack.
2. popReturnRequest() – Process and remove the latest request.
3. peekLatestRequest() – View the most recent request awaiting review.
4. displayAllRequests() – Display all unprocessed requests.

This simulation helps analyze how LIFO operations assist in managing real-time customer service workflows.

Q#2: Browser History Navigation Manager (Dynamic Stack Implementation)

Modern browsers (e.g., Google Chrome, Mozilla Firefox) maintain a navigation history that allows users to go back and forth between visited websites. This navigation history internally functions as a stack.

Implement a dynamic stack (linked-list-based) to simulate a simplified browser history system. Each node represents a webpage containing:
• Page Title
• URL
• Time Visited

Operations:
1. visitPage(url)
2. goBack()
3. viewCurrentPage()
4. displayHistory()

This case study demonstrates how browsers use stack principles to manage navigation flow efficiently.

Q#3: Expression Evaluation in a Scientific Calculator (Application of Stack)

A scientific calculator application developed for engineers must evaluate mathematical expressions entered in postfix (Reverse Polish) notation. Each expression may contain:
• Single- or multi-digit operands (non-negative integers)
• Arithmetic operators: +, -, *, /

Design a stack-based evaluator that computes the result of such expressions.
Example: Input → 23-4+567*+* | Output → 141

The evaluator should process each character sequentially, pushing operands and applying operators as they appear.

Q#4: Warehouse Inventory Stacking Simulator (Application of Stack)

A logistics company uses robotic arms to stack and unstack boxes in a warehouse. To maintain safety, no heavier box may be placed on a lighter one. When a new box arrives:
• If it's lighter than the top box → push it.
• If it's heavier → repeatedly pop boxes until a heavier or equal box is found, then push it.

Display the final stack configuration from top (lightest) to bottom (heaviest). This models warehouse stacking algorithms used in robotic sorting.

Q#5: Airport Shuttle Ticketing Queue (Static Queue Implementation)

At large airports, a shuttle service transports passengers between terminals. Passengers queue up to buy tickets following the first-come, first-served rule.

Design a static queue (array-based) that manages ticket sales. Each passenger record includes:
• Name
• Destination Terminal
• Ticket Type (VIP/Regular)

Operations:
1. enqueuePassenger()
2. dequeuePassenger()
3. displayQueue()
4. isFull()/isEmpty()

This queue system mirrors real airport automation software used for passenger flow management.

Q#6: Hospital Appointment Scheduling System (Dynamic Queue Implementation)

Hospitals often need to dynamically manage patient appointments as new cases arrive. Develop a dynamic queue (linked-list-based) for scheduling patient consultations.

Each record stores:
• Patient Name
• Patient ID
• Doctor's Name
• Appointment Time

Operations:
1. addPatient()
2. callNextPatient()
3. displayAllAppointments()

This simulation reflects real-time queuing logic used in hospital management systems to improve patient flow.

Q#7: Smart Parking Lot Management System (Circular Queue Application)

A smart parking lot with fixed slots uses a circular queue to track parked cars and efficiently reuse empty slots.

Each car record includes:
• Car Number
• Arrival Time
• Slot Number

Operations:
1. parkCar()
2. removeCar()
3. displayParkingStatus()

This case study represents how smart parking systems optimize limited parking spaces using circular data structures.

Q#8: Train Compartment Reservation System (Double-Ended Queue Application)

The national railway's reservation system must handle both VIP and regular passengers. VIP passengers get front-end bookings, while regular passengers are queued from the rear end.

Design a double-ended circular queue (deque) to manage compartment reservations. Each record includes:

• Passenger Name
• Category (VIP/Regular)
• Seat Number

Operations:
1. bookAtFront()
2. bookAtRear()
3. cancelFromFront()/cancelFromRear()
4. displayAllReservations()

This models how transportation systems dynamically prioritize customers without disrupting existing orders.

Q#9: Emergency Room Patient Management (Priority Queue Application)

In a hospital Emergency Room (ER), patients are triaged based on urgency levels (1=Critical, 4=Low). Implement a priority queue that maintains patients according to priority.

Each record includes:
• Patient ID
• Name
• Condition Description
• Priority Level

Simulate arrivals in order 4,1,3,2 and ensure they are treated in the order 1→2→3→4. This demonstrates how hospital triage systems manage patient queues for optimal treatment sequencing.

Q#10: Food Delivery Dispatch System (Application of Queue)

A food delivery service (like Foodpanda) must assign orders to riders in real time. Each order includes:
• Order ID
• Restaurant Name
• Customer Address
• Item List
• Preparation Time
• Order Time

Operations:
1. addOrder() – Insert at front if quick items only, else at rear.
2. dispatchOrder()
3. viewNextOrder()
4. displayAllOrders()
5. cancelOrder(orderID)

This models the dispatch logic of real delivery systems, balancing quick orders and high-priority deliveries using queue operations.