CAM2003C - Data Structures and Algorithms with C and C++

Lab Exercise -7: Queue Data Structure Implementation using Array and Linked Lists and Applications of Queue ADT

Practical Questions on Queue

Objective

- To understand the **concept, implementation, and applications** of queues.
- To implement **linear, circular, priority, and multiple queues**.
- To analyze **time and space complexities** of different queue operations.

Experiment 1: Linear Queue using Array

Objectives:

- Implement a **simple linear queue**.
- Perform **enqueue**, **dequeue**, **peek**, **isEmpty**, and **isFull** operations.

Tasks:

1. Define an array queue[MAX] and variables front and rear.
2. Implement **enqueue** operation with overflow check.
3. Implement **dequeue** operation with underflow check.
4. Implement **peek/front** operation.
5. Implement **display** function to show all elements.

Expected Outcome:

- Students will understand **FIFO behavior** and the limitations of linear queue (space wastage).

Experiment 2: Queue using Linked List

Objectives:

- Implement a queue using **singly linked list**.
- Understand **dynamic memory allocation** and **pointer management**.

Tasks:

1. Create a **node structure** with data and next.
2. Implement **enqueue** at the **rear**.
3. Implement **dequeue** at the **front**.
4. Implement **peek** operation.
5. Display queue elements.

Expected Outcome:

- Queue size can grow **dynamically**.
- Students will understand **difference between array-based and linked-list queue**.

Experiment 3: Circular Queue using Array

Objectives:

- Implement a **circular queue** to overcome the wastage of linear queue.

Tasks:

1. Define array cq[MAX], front, and rear.
2. Implement **enqueue** with circular increment (rear + 1) % MAX.
3. Implement **dequeue** with circular increment (front + 1) % MAX.
4. Implement **display** function handling circular nature.

Expected Outcome:

- Efficient memory utilization in queues.
- Understand **modulus operation** for circular behavior.

Experiment 4: Double-Ended Queue (Deque)

Objectives:

- Implement **deque** using array.
- Insert and delete elements from **both ends**.

Tasks:

1. Define front and rear pointers.
2. Implement **insertFront**, **insertRear**, **deleteFront**, **deleteRear**.
3. Display queue elements after each operation.

Expected Outcome:

- Students will learn **flexible queue operations**.
- Real-world analogy: Undo/Redo stack in editors.

Experiment 5: Priority Queue using Array

Objectives:

- Implement a **priority queue** using array.
- Understand **priority-based element selection**.

Tasks:

1. Define an array of Element {data, priority}.
2. Implement **enqueue** (unsorted array → O(1)).
3. Implement **dequeue** (find highest priority → O(n)).
4. Implement **peek** operation.
5. Display queue with priorities.

Expected Outcome:

- Understand **priority scheduling**, like CPU or OS jobs.
- Compare **unsorted vs sorted array** implementation.

Experiment 6: Queue using Two Stacks

Objectives:

- Implement a queue using **two stacks**.
- Compare **enqueue-costly** vs **dequeue-costly** methods.

Tasks:

1. Define two stacks s1 and s2.
2. Implement **enqueue-costly method**: push O(n), pop O(1).
3. Implement **dequeue-costly method**: push O(1), pop O(n).
4. Test with multiple enqueue and dequeue operations.

Expected Outcome:

- Understand **queue-stack relationship** and **algorithmic trade-offs**.

Experiment 7: Stack using Two Queues

Objectives:

- Implement a stack using **two queues**.
- Compare **push-costly** vs **pop-costly** implementations.

Tasks:

1. Define two queues `q1` and `q2`.
2. Implement **push-costly method**: push O(n), pop O(1).
3. Implement **pop-costly method**: push O(1), pop O(n).
4. Test LIFO behavior using different sequences.

Expected Outcome:

- Understand **stack-queue relationship**.
- Learn **algorithmic trade-offs** between methods.

Experiment 8: Multiple Queues in a Single Array

Objectives:

- Implement **two queues in one array**.
- Efficient memory usage by sharing array space.

Tasks:

1. Define `arr[MAX]`, `front1`, `rear1`, `front2`, `rear2`.
2. Queue1 grows **left → right**, Queue2 grows **right → left**.
3. Implement **enqueue** and **dequeue** for both queues.
4. Display elements of both queues.

Expected Outcome:

- Learn **memory-efficient queue design**.
- Understand **overflow conditions** for multiple queues.

Experiment 9: Applications of Queue

Objectives:

- Implement **real-world scenarios** using queue.

Tasks:

1. **CPU Scheduling Simulation**: Queue stores process IDs, simulate FCFS scheduling.
2. **Print Spooler**: Queue stores print jobs, simulate printing in order.
3. **Customer Service**: Priority queue for VIP and normal customers.

Expected Outcome:

- Understand **practical applications** of different queue types.
- Learn **priority handling in real systems**.

Lab Deliverables

1. Source code for each experiment.
2. Input/output screenshots.
3. Complexity analysis of each operation.