University of Engineering and Technology Taxila
Computer Engineering Department

Data Structure and Algorithm
Lab Manual # 08

Lab Instructor: Engr. Shahryar Khan

University of Engineering and Technology Taxila
Computer Engineering Department

Lab Title: **Practicing Queues for Efficient Data Processing**

Lab Overview

This lab focuses on understanding and implementing Queue data structures and their real-world applications. Students will explore different implementations of queues, including list-based queues, stack-based queues, and node-based queues (linked list queues). The lab also covers FIFO (First-In-First-Out) processing, queue operations (enqueue and dequeue), and applications like task scheduling, print queue management, and media player playlists.
By the end of this lab, students will be able to implement and manipulate queue structures efficiently to solve practical computing problems.

Lab Objectives

By the end of this lab, students will be able to:

- Understand and implement Queue operations (Enqueue, Dequeue, Peek, Size).
- Implement List-based, Stack-based, and Node-based (linked list) queues in Python.
- Analyze the advantages and disadvantages of different queue implementations.
- Explore real-world applications of queues, such as process scheduling, media players, and network packet handling.
- Apply queue concepts to real-world problem-solving scenarios

Note: For the guided tasks type the code yourself.

Guided Tasks

Task 1: Implementing a List-Based Queue

Objective: Implement a basic queue using Python's built-in list.

Implementation:

```
class ListQueue:
    def __init__(self):
        self.items = []

    def enqueue(self, data):
        """Add an item to the queue (end of list)."""
        self.items.insert(0, data)

    def dequeue(self):
        """Remove and return the first item from the queue."""
        if self.is_empty():
            print("Queue is empty!")
            return None
        return self.items.pop()

    def peek(self):
        """Return the first element without removing it."""
        return self.items[-1] if self.items else None

    def size(self):
        """Return the size of the queue."""
        return len(self.items)

    def is_empty(self):
        """Check if queue is empty."""
        return len(self.items) == 0

# Example Usage
queue = ListQueue()
queue.enqueue("Task 1")
queue.enqueue("Task 2")
print("First item:", queue.peek())  # Task 1
print("Dequeued:", queue.dequeue())  # Task 1
print("Queue size:", queue.size())  # 1
```

Task 2: Implementing a Stack-Based Queue

Objective: Implement a queue using two stacks (list-based).

Implementation:

```
class StackQueue:
    def __init__(self):
        self.inbound_stack = []
        self.outbound_stack = []

    def enqueue(self, data):
        """Push data to inbound stack."""
        self.inbound_stack.append(data)

    def dequeue(self):
        """Move elements from inbound to outbound stack, then pop."""
        if not self.outbound_stack:
            while self.inbound_stack:
                self.outbound_stack.append(self.inbound_stack.pop())
        return self.outbound_stack.pop() if self.outbound_stack else None

# Example Usage
sq = StackQueue()
sq.enqueue(5)
sq.enqueue(10)
sq.enqueue(15)
print("Dequeued:", sq.dequeue())  # 5
print("Dequeued:", sq.dequeue())  # 10
```

Task 3: Implementing a Node-Based Queue (Linked List)

Objective: Implement a queue using a **doubly linked list**.

Implementation:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedQueue:
    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0

    def enqueue(self, data):
        """Add an element to the tail of the queue."""
        new_node = Node(data)
        if self.tail:
            self.tail.next = new_node
```

```
        self.tail = new_node
        if self.head is None:
            self.head = new_node
        self.size += 1

    def dequeue(self):
        """Remove the front node."""
        if self.head is None:
            return None
        removed_data = self.head.data
        self.head = self.head.next
        if self.head is None:
            self.tail = None
        self.size -= 1
        return removed_data

# Example Usage
lq = LinkedQueue()
lq.enqueue(100)
lq.enqueue(200)
print("Dequeued:", lq.dequeue())  # 100
```

Exercise Questions

Easy Problems

1. **Basic Queue Operations:** Implement enqueue, dequeue, peek, and size functions.
2. **Reverse a Queue:** Implement a function to reverse a queue using a stack.
3. **Check Palindrome using Queue:** Use a queue to check if a string is a palindrome.
4. **Queue-based Task Manager:** Implement a simple task manager using a queue.
5. **Print Jobs Simulation:** Simulate print job handling using a queue.

Intermediate Problems

1. **Call Center Simulation:** Implement a queue where customer service calls are answered in FIFO order.
2. **CPU Task Scheduling:** Implement Round Robin scheduling for CPU tasks using a queue.
3. **Message Queue System:** Implement a simple message-passing queue system between users.
4. **Queue-based Chat System:** Implement a chat message queue where messages are displayed in order.

5. **Ride-Sharing Queue:** Simulate a queue system where passengers are assigned to rides based on first-come, first-served.

Advanced Problems

1. **Facebook Messenger Chat Queue:** Implement a queue system where chat messages are stored and displayed in order.
2. **Spotify Playlist Queue:** Implement a **circular queue** to cycle through songs in a playlist.
3. **Operating System Process Queue:** Simulate how an operating system manages processes using a priority queue.
4. **Network Packet Handling Queue:** Implement a queue to process packets sent over a network.
5. **AI Task Processing Queue:** Implement a queue that assigns AI processing tasks to available GPU resources.