Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report 6

Obtained Marks: Not Evaluated Total Marks: 8

**Marks Distribution:**Total Lab Activity Marks:4   
Total Lab Report Marks: 4

Lab 5

**Guided Tasks (Circular Linked List)**

**Task 1: Implementing a Circular Linked List (CLL)**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output:**

****

**Task 2: Instagram Story Viewer using a Circular Linked List**

****

**A screen shot of a computer

AI-generated content may be incorrect.**

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**EXERCISE**

**Easy Problems**

**1. CLL Traversal**

Implement a Circular Linked List and traverse it in a loop.

**class Node:**

**def \_\_init\_\_(self, data):**

**self.data = data**

**self.next = None**

**class CircularLinkedList:**

**def \_\_init\_\_(self):**

**self.head = None**

**def append(self, data):**

**new\_node = Node(data)**

**if not self.head:**

**self.head = new\_node**

**new\_node.next = self.head**

**else:**

**temp = self.head**

**while temp.next != self.head:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.next = self.head**

**def traverse(self):**

**if not self.head:**

**print("List is empty")**

**return**

**temp = self.head**

**while True:**

**print(temp.data, end=" -> ")**

**temp = temp.next**

**if temp == self.head:**

**break**

**print("(back to head)")**

**# Usage**

**cll = CircularLinkedList()**

**cll.append(1)**

**cll.append(2)**

**cll.append(3)**

**cll.traverse()**

**Output:**

****

**2. CLL Deletion**

Implement a method to delete a node in Circular Linked List.

**class Node:**

**def \_\_init\_\_(self, data):**

**self.data = data**

**self.next = None**

**class CircularLinkedList:**

**def \_\_init\_\_(self):**

**self.head = None**

**def append(self, data):**

**new\_node = Node(data)**

**if not self.head:**

**self.head = new\_node**

**new\_node.next = self.head**

**else:**

**temp = self.head**

**while temp.next != self.head:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.next = self.head**

**def delete\_node(self, key):**

**if not self.head:**

**print("List is empty")**

**return**

**temp = self.head**

**prev = None**

**while True:**

**if temp.data == key:**

**if prev:**

**prev.next = temp.next**

**else:**

**last = self.head**

**while last.next != self.head:**

**last = last.next**

**self.head = temp.next**

**last.next = self.head**

**return**

**prev, temp = temp, temp.next**

**if temp == self.head:**

**break**

**print("Node not found")**

**def traverse(self):**

**if not self.head:**

**print("List is empty")**

**return**

**temp = self.head**

**while True:**

**print(temp.data, end=" -> ")**

**temp = temp.next**

**if temp == self.head:**

**break**

**print("(back to head)")**

**# Usage**

**cll = CircularLinkedList()**

**cll.append(1)**

**cll.append(2)**

**cll.append(3)**

**cll.traverse()**

**cll.delete\_node(2)**

**cll.traverse()**

**Output:**

****

**Intermediate Problems**

**1. Circular Scheduling System (CLL)**

Implement a task scheduling system where tasks repeat cyclically using a Circular Linked List.

**class Node:**

**def \_\_init\_\_(self, task):**

**self.task = task**

**self.next = None**

**class TaskScheduler:**

**def \_\_init\_\_(self):**

**self.head = None**

**def add\_task(self, task):**

**new\_node = Node(task)**

**if not self.head:**

**self.head = new\_node**

**new\_node.next = self.head**

**else:**

**temp = self.head**

**while temp.next != self.head:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.next = self.head**

**def execute\_tasks(self, cycles=2):**

**if not self.head:**

**print("No tasks to execute.")**

**return**

**temp = self.head**

**for \_ in range(cycles):**

**print(f"Executing: {temp.task}")**

**temp = temp.next**

**print("(Tasks repeated)")**

**# Usage**

**scheduler = TaskScheduler()**

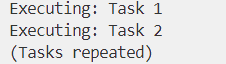
**scheduler.add\_task("Task 1")**

**scheduler.add\_task("Task 2")**

**scheduler.add\_task("Task 3")**

**scheduler.execute\_tasks()**

**Output:**

****

**2. Round Robin CPU Scheduling (CLL)**

Simulate Round Robin CPU scheduling using a Circular Linked List.

**class Node:**

**def \_\_init\_\_(self, process, time):**

**self.process = process**

**self.time = time**

**self.next = None**

**class CPU\_Scheduler:**

**def \_\_init\_\_(self):**

**self.head = None**

**def add\_process(self, process, time):**

**new\_node = Node(process, time)**

**if not self.head:**

**self.head = new\_node**

**new\_node.next = self.head**

**else:**

**temp = self.head**

**while temp.next != self.head:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.next = self.head**

**def execute(self, quantum=3):**

**if not self.head:**

**print("No processes to execute.")**

**return**

**temp = self.head**

**while True:**

**if temp.time > 0:**

**execute\_time = min(temp.time, quantum)**

**temp.time -= execute\_time**

**print(f"Executing {temp.process} for {execute\_time} units")**

**if temp.time == 0:**

**print(f"{temp.process} completed.")**

**temp = temp.next**

**if temp == self.head and all(node.time == 0 for node in self.\_iter\_nodes()):**

**break**

**def \_iter\_nodes(self):**

**temp = self.head**

**if not temp:**

**return**

**while True:**

**yield temp**

**temp = temp.next**

**if temp == self.head:**

**break**

**# Usage**

**scheduler = CPU\_Scheduler()**

**scheduler.add\_process("P1", 5)**

**scheduler.add\_process("P2", 7)**

**scheduler.add\_process("P3", 4)**

**scheduler.execute()**

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**