Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report 5

Obtained Marks: Not Evaluated Total Marks: 8

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

Lab 5

**Guided Tasks**

**Task 1:** Implementing a Doubly Linked List (DLL)

****

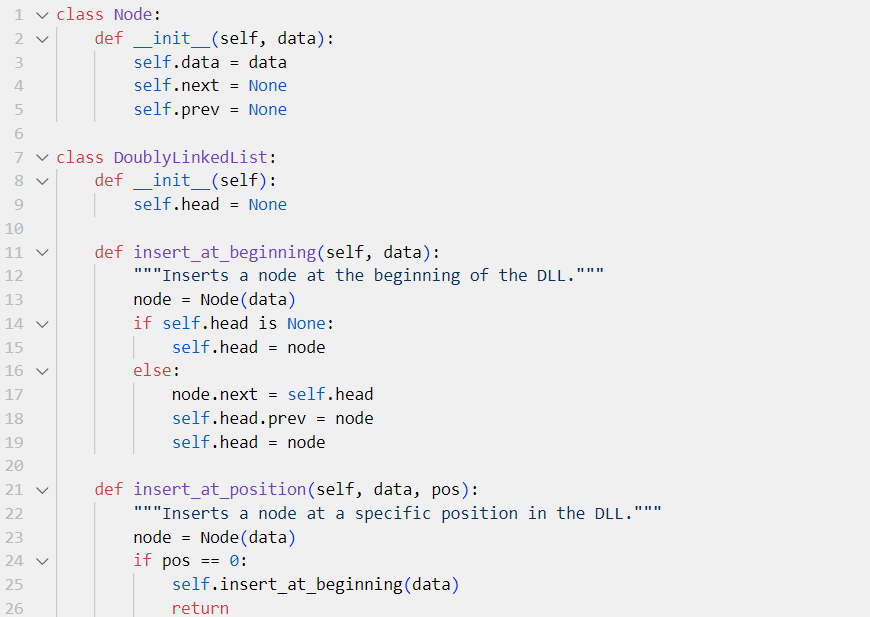
**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

****

**Task 2:** Insertion Operations in DLL

****

**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 3: 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 4: Music Playlist System using a Doubly Linked List**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Task 5: 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-DLL Basic Operations**

Implement a class for Doubly Linked List that supports append, display, and delete from start.

**class Node:**

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

**self.data = data**

**self.prev = None**

**self.next = None**

**class DoublyLinkedList:**

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

**self.head = None**

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

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

**if not self.head:**

**self.head = new\_node**

**return**

**temp = self.head**

**while temp.next:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.prev = temp**

**def display(self):**

**temp = self.head**

**while temp:**

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

**temp = temp.next**

**print("None")**

**def delete\_from\_start(self):**

**if not self.head:**

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

**return**

**self.head = self.head.next**

**if self.head:**

**self.head.prev = None**

**# Usage**

**dll = DoublyLinkedList()**

**dll.append(10)**

**dll.append(20)**

**dll.append(30)**

**dll.display()**

**dll.delete\_from\_start()**

**dll.display()**

**Output:**

****

**2. 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:**

****

**3. DLL Reverse Traversal**

Implement a method to print a DLL in reverse order.

**class Node:**

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

**self.data = data**

**self.next = None**

**self.prev = None**

**class DoublyLinkedList:**

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

**self.head = None**

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

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

**if not self.head:**

**self.head = new\_node**

**return**

**temp = self.head**

**while temp.next:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.prev = temp**

**def reverse\_traverse(self):**

**temp = self.head**

**if not temp:**

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

**return**

**while temp.next:**

**temp = temp.next**

**while temp:**

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

**temp = temp.prev**

**print("None")**

**# Usage**

**dll = DoublyLinkedList()**

**dll.append(10)**

**dll.append(20)**

**dll.append(30)**

**dll.reverse\_traverse()**

**Output:**

****

**4. 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:**

****

**5. DLL Length Calculation**

Implement a function that returns the length of a DLL.

**class Node:**

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

**self.data = data**

**self.next = None**

**self.prev = None**

**class DoublyLinkedList:**

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

**self.head = None**

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

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

**if not self.head:**

**self.head = new\_node**

**return**

**temp = self.head**

**while temp.next:**

**temp = temp.next**

**temp.next = new\_node**

**new\_node.prev = temp**

**def get\_length(self):**

**count = 0**

**temp = self.head**

**while temp:**

**count += 1**

**temp = temp.next**

**return count**

**# Usage**

**dll = DoublyLinkedList()**

**dll.append(10)**

**dll.append(20)**

**dll.append(30)**

**print("Length:", dll.get\_length())**

**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()**

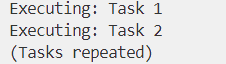
**scheduler.add\_task("Task 1")**

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

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

**scheduler.execute\_tasks()**

**Output:**

****

**2. Game Leaderboard (DLL)**

Implement a leaderboard where scores are stored in a Doubly Linked List, sorted by highest score.

**class Node:**

**def \_\_init\_\_(self, name, score):**

**self.name = name**

**self.score = score**

**self.next = None**

**self.prev = None**

**class Leaderboard:**

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

**self.head = None**

**def add\_score(self, name, score):**

**new\_node = Node(name, score)**

**if not self.head or self.head.score < score:**

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

**if self.head:**

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

**self.head = new\_node**

**return**

**temp = self.head**

**while temp.next and temp.next.score >= score:**

**temp = temp.next**

**new\_node.next = temp.next**

**if temp.next:**

**temp.next.prev = new\_node**

**temp.next = new\_node**

**new\_node.prev = temp**

**def display\_leaderboard(self):**

**temp = self.head**

**while temp:**

**print(f"{temp.name}: {temp.score}")**

**temp = temp.next**

**# Usage**

**board = Leaderboard()**

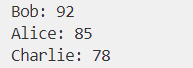
**board.add\_score("Alice", 85)**

**board.add\_score("Bob", 92)**

**board.add\_score("Charlie", 78)**

**board.display\_leaderboard()**

**Output:**

****

**3. 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.**

**Advanced Problems**

**1. Facebook Messenger Chat History (DLL)**

Implement a chat history feature using a Doubly Linked List to navigate through messages.

**class Node:**

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

**self.message = message**

**self.next = None**

**self.prev = None**

**class ChatHistory:**

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

**self.head = None**

**self.tail = None**

**def add\_message(self, message):**

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

**if not self.head:**

**self.head = new\_node**

**self.tail = new\_node**

**else:**

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

**new\_node.prev = self.tail**

**self.tail = new\_node**

**def show\_history(self):**

**temp = self.tail**

**while temp:**

**print(temp.message)**

**temp = temp.prev**

**# Usage**

**chat = ChatHistory()**

**chat.add\_message("Hello")**

**chat.add\_message("How are you?")**

**chat.add\_message("I'm good, thanks!")**

**chat.show\_history()**

**Output:**

**A close up of words

AI-generated content may be incorrect.**

**2. Undo/Redo System (DLL)**

Implement an Undo/Redo system for a text editor using Doubly Linked Lists.

**class Node:**

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

**self.text = text**

**self.next = None**

**self.prev = None**

**class TextEditor:**

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

**self.head = None**

**self.current = None**

**def write(self, text):**

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

**if not self.head:**

**self.head = new\_node**

**self.current = new\_node**

**else:**

**new\_node.prev = self.current**

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

**self.current = new\_node**

**def undo(self):**

**if self.current and self.current.prev:**

**self.current = self.current.prev**

**print("Current Text:", self.current.text if self.current else "Empty")**

**def redo(self):**

**if self.current and self.current.next:**

**self.current = self.current.next**

**print("Current Text:", self.current.text if self.current else "Empty")**

**# Usage**

**editor = TextEditor()**

**editor.write("Hello")**

**editor.write("World")**

**editor.undo()**

**editor.redo()**

**Output:**

****

**3. Browser History Navigation (DLL)**

Implement forward and backward navigation in a web browser using a Doubly Linked List.

**class Node:**

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

**self.url = url**

**self.next = None**

**self.prev = None**

**class BrowserHistory:**

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

**self.current = None**

**def visit(self, url):**

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

**if not self.current:**

**self.current = new\_node**

**else:**

**new\_node.prev = self.current**

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

**self.current = new\_node**

**def back(self):**

**if self.current and self.current.prev:**

**self.current = self.current.prev**

**print("Current Page:", self.current.url if self.current else "No history")**

**def forward(self):**

**if self.current and self.current.next:**

**self.current = self.current.next**

**print("Current Page:", self.current.url if self.current else "No forward history")**

**# Usage**

**browser = BrowserHistory()**

**browser.visit("google.com")**

**browser.visit("facebook.com")**

**browser.back()**

**browser.forward()**

**Output:**

****