Experiment 4

Student Name: ISHITA NISHANT **UID: 23BCS11354**
Branch: BE CSE **Section/Group: KRG-2A**
Semester: 5 **DateofPerformance:18/8/2025**
Subject Name: DAA **Subject Code:23CSH-301**

1. **Aim:** Apply the concept of Linked list and write code to Insert and Delete an element at the beginning and append in Doubly and Circular Linked List.

2. **Objective:** To learn the concept of Doubly and Circular Linked List

3. **Implementation/Code:**

```
#include <iostream>

using namespace std;

// ------------------- Doubly Linked List -------------------

class DLLNode {

public:

int data;

DLLNode* prev;

DLLNode* next;

DLLNode(int val) {

data = val;

prev = next = nullptr;
```

```
}
};
class DoublyLinkedList {
DLLNode* head;
DLLNode* tail;
public:
DoublyLinkedList() {
head = tail = nullptr;
}
void insertAtBeginning(int val) {
DLLNode* newNode = new DLLNode(val);
if (!head) {
head = tail = newNode;
} else {
newNode->next = head;
head->prev = newNode;
head = newNode;
}
}
void insertAtEnd(int val) {
DLLNode* newNode = new DLLNode(val);
```

```
if (!tail) {

head = tail = newNode;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

void deleteAtBeginning() {

if (!head) return;

DLLNode* temp = head;

head = head->next;

if (head) head->prev = nullptr;

else tail = nullptr;

delete temp;

}

void deleteAtEnd() {

if (!tail) return;

DLLNode* temp = tail;

tail = tail->prev;

if (tail) tail->next = nullptr;
```

```
else head = nullptr;

delete temp;

}

void display() {

DLLNode* temp = head;

cout << "DLL: ";

while (temp) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

// ------------------- Circular Linked List -------------------

class CLLNode {

public:

int data;

CLLNode* next;

CLLNode(int val) {

data = val;

next = nullptr;
```

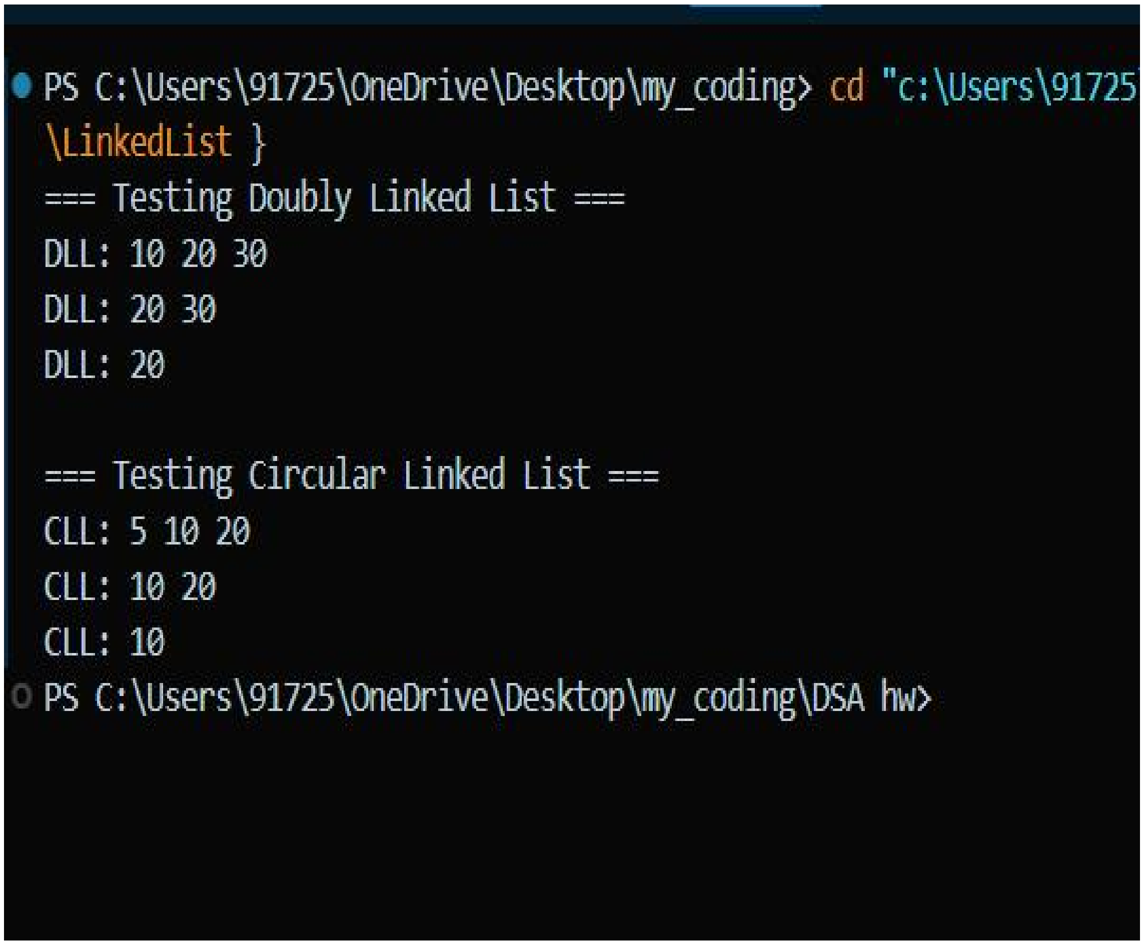
```
}
};
class CircularLinkedList {
CLLNode* head;
CLLNode* tail;
public:
CircularLinkedList() {
head = tail = nullptr;
}
void insertAtBeginning(int val) {
CLLNode* newNode = new CLLNode(val);
if (!head) {
head = tail = newNode;
newNode->next = head;
} else {
newNode->next = head;
head = newNode;
tail->next = head;
}
}
void insertAtEnd(int val) {
```

```
CLLNode* newNode = new CLLNode(val);

if (!tail) {

head = tail = newNode;

newNode->next = head;

} else {

tail->next = newNode;

tail = newNode;

tail->next = head;

}

}

void deleteAtBeginning() {

if (!head) return;

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

CLLNode* temp = head;

head = head->next;

tail->next = head;

delete temp;
```

```
}
void deleteAtEnd() {
if (!head) return;
if (head == tail) {
delete head;
head = tail = nullptr;
return;
}
CLLNode* temp = head;
while (temp->next != tail) {
temp = temp->next;
}
temp->next = head;
delete tail;
tail = temp;
}
void display() {
if (!head) {
cout << "CLL: (empty)" << endl;
return;
}
```

```
CLLNode* temp = head;

cout << "CLL: ";

do {

cout << temp->data << " ";

temp = temp->next;

} while (temp != head);

cout << endl;

}

};

// ------------------ Main ------------------

int main() {

DoublyLinkedList dll;

CircularLinkedList cll;

cout << "=== Testing Doubly Linked List ===" << endl;

dll.insertAtBeginning(10);

dll.insertAtEnd(20);

dll.insertAtEnd(30);

dll.display(); // 10 20 30

dll.deleteAtBeginning();

dll.display(); // 20 30

dll.deleteAtEnd();
dll.display(); // 20

cout << "\n=== Testing Circular Linked List ===" << endl;

cll.insertAtEnd(10);

cll.insertAtEnd(20);

cll.insertAtBeginning(5);

cll.display(); // 5 10 20

cll.deleteAtBeginning();

cll.display(); // 10 20

cll.deleteAtEnd();

cll.display(); // 10

return 0;

}
```

4. Output



```
PS C:\Users\91725\OneDrive\Desktop\my_coding> cd "c:\Users\91725
\LinkedList }
=== Testing Doubly Linked List ===
DLL: 10 20 30
DLL: 20 30
DLL: 20

=== Testing Circular Linked List ===
CLL: 5 10 20
CLL: 10 20
CLL: 10
PS C:\Users\91725\OneDrive\Desktop\my_coding\DSA hw>
```

5. Learning Outcome

- Understand the structure and operations of **Doubly** and **Circular Linked Lists**.
- Gain proficiency in **pointer manipulation** for insertion and deletion.
- Learn to handle **edge cases** (empty list, single-node list).