Experiment-4

Student Name: Tejinderpal Singh
UID: 23BCS80343
Branch: BE-CSE
Section/Group: 23BCS_KRG_3A
Semester: 5th
Subject Code: 23CSH-301
Subject Name: DAA

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

2. **Objective:** To understand and implement insertion and deletion operations at both the beginning and the end of doubly linked lists and circular singly linked lists using C++ classes. The objective is to help students learn pointer manipulation, dynamic memory management, bi-directional traversal in doubly linked lists, and circular node connections in circular linked lists, thereby strengthening their understanding of dynamic data structures and their real-time applications.

3. **Procedure:**

A. For Doubly Linked List:

1. Start.
2. Define a class DLL with data, prev, and next pointers.
3. Create a DLL class object with head pointer initialized to NULL.
4. Define insertAtBegin():
   - Create a new node.
   - If the list is empty, set both head and tail to the new node.
   - Otherwise, link the new node with the current head and update head.
5. Define insertAtEnd():
   - Create a new node.
   - If the list is empty, set both head and tail to the new node.
   - Otherwise, link the new node with the current tail and update tail.
6. Define deleteAtBegin():
   - If the list is empty, display a message and return.
   - If only one node exists, delete it and set head and tail to NULL.
   - Otherwise, move head to the next node and update pointers.
7. Define deleteAtEnd():
   - If the list is empty, display a message and return.

- If only one node exists, delete it and set head and tail to NULL.
- Otherwise, move tail to the previous node and update pointers.

8. Define display() to traverse from head and print each node's data.
9. End.

B. For Circular Linked List:

1. Start.
2. Define a class CLL with data and next pointer.
3. Create a CLL class object with tail pointer initialized to NULL.
4. Define insertAtBegin():
   - Create a new node.
   - If the list is empty, point the node to itself and set as tail.
   - Otherwise, link the new node before the head and update tail's next.
5. Define insertAtEnd():
   - Create a new node.
   - If the list is empty, point the node to itself and set as tail.
   - Otherwise, link the new node after tail and update tail.
6. Define deleteAtBegin():
   - If the list is empty, display a message and return.
   - If only one node exists, delete it and set tail to NULL.
   - Otherwise, bypass the head node and update tail's next pointer.
7. Define deleteAtEnd():
   - If the list is empty, display a message and return.
   - If only one node exists, delete it and set tail to NULL.
   - Otherwise, traverse to the node before tail, update it as the new tail, and link it to head.
8. Define display():
   - If the list is empty, display a message.
   - Otherwise, start from tail->next and traverse until the starting node is reached again.
9. End.

4. Code:

```
#include  <iostream>
using namespace std;
class DLL {
public:
    int data;
    DLL* next;
    DLL* prev;
    DLL* head;
    DLL() {
        head = NULL;
        next = NULL;
        prev = NULL;
    }
    void insertAtBegin(int d) {
        DLL* nn = new DLL;
        nn->data = d;
        nn->next = head;
        nn->prev = NULL;
        if (head != NULL)
            head->prev = nn;
        head = nn;
    }
    void insertAtEnd(int d) {
        DLL* nn = new DLL;
        nn->data = d;
        nn->next  =  NULL;
        if (head == NULL) {
            nn->prev = NULL;
            head = nn;
            return;
        }
        DLL* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = nn;
        nn->prev = temp;
    }
```

```
    void deleteAtBegin() {
        if (head == NULL) return;
        DLL* temp = head;
        head = head->next;
        if (head != NULL)
            head->prev = NULL;
        delete temp;
    }
    void deleteAtEnd() {
        if (head == NULL) return;
        DLL* temp = head;
        if (head->next == NULL) {
            delete head;
            head = NULL;
            return;
        }
        while (temp->next != NULL)
            temp = temp->next;
        temp->prev->next = NULL;
        delete temp;
    }
    void display() {
        DLL* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
class CLL {
public:
    int data;
    CLL* next;
    CLL* tail;
    CLL() {
        tail = NULL;
        next = NULL;
```

```
}
void insertAtBegin(int d) {
   CLL* nn = new CLL;
   nn->data = d;
   if (tail == NULL) {
      tail = nn;
      tail->next = tail;
   } else {
      nn->next = tail->next;
      tail->next = nn;
   }
}
void insertAtEnd(int d) {
   CLL* nn = new CLL;
   nn->data = d;
   if (tail == NULL) {
      tail = nn;
      tail->next = tail;
   } else {
      nn->next = tail->next;
      tail->next = nn;
      tail = nn;
   }
}
void deleteAtBegin() {
   if (tail == NULL) return;
   CLL* head = tail->next;
   if (head == tail) {
      delete  head;
      tail = NULL;
   } else {
      tail->next = head->next;
      delete head;
   }
}
void deleteAtEnd() {
   if (tail == NULL) return;
   CLL* temp = tail->next;
   if (temp == tail) {
```

```
            delete tail;
            tail = NULL;
        } else {
            while (temp->next != tail)
                temp = temp->next;
            temp->next = tail->next;
            delete tail;
            tail = temp;
        }
    }
    void display() {
        if (tail == NULL) {
            cout << "List is empty\n";
            return;
        }
        CLL* temp = tail->next;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != tail->next);
        cout << endl;
    }
};
int main() {
    DLL  o1;
    CLL o2;
    cout << "Doubly Linked List:"<<endl;
    o1.insertAtBegin(10);
    o1.insertAtEnd(20);
    o1.insertAtBegin(5);
    o1.display();
    o1.deleteAtBegin();
    o1.deleteAtEnd();
    o1.display();
    cout << "\nCircular Linked List:\n";
    o2.insertAtBegin(10);
    o2.insertAtEnd(20);
    o2.insertAtBegin(5);
    o2.display();
```

```
    o2.deleteAtBegin();
    o2.deleteAtEnd();
    o2.display();
    return 0;
}
```

5. Observations:

```
Doubly Linked List:
5 10 20
10

Circular Linked List:
5 10 20
10
```

6. Time Complexity:

Operation	Doubly Linked List	Cicular Linked List
Insert at Begin	O(1)	O(1)
Insertion at End	O(n)	O(1)
Delete at Begin	O(1)	O(1)
Deletion at End	O(n)	O(n)
Traversal	O(n)	O(n)

7. Learning Outcome:

- Learned how to implement and manipulate doubly and circular linked lists using classes in C++.
- Understood the structural differences between singly, doubly, and circular linked lists.
- Gained practical experience in inserting and deleting nodes at both beginning and end of the list.
- Strengthened understanding of pointer manipulation and dynamic memory allocation in linked list operations.
- Practiced modular programming by encapsulating linked list logic within class functions.