Experiment:1.4

Student Name: Virat Samdarshi **UID: 22BCS12648**
Branch: B.E CSE **Section/Group:IOT-627-B**
Semester:5th **DateofPerformance:07-08-24**
Subject Name:DAAlab **SubjectCode:22CSH-311**

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:** The objective of this experiment is to implement and demonstrate the basic operations—insertion and deletion—at both the beginning and end of Doubly Linked Lists and Circular Linked Lists, showcasing how these structures can be efficiently managed and manipulated.

3. **Implementation/Code:**

Insertion and Deletion of the element at the beginning and at the end in Doubly Linked List

```
#include <iostream>
using namespace std;

class DoublyNode {
public:
    int data;
    DoublyNode* prev;
    DoublyNode* next;

    DoublyNode(int data) : data(data), prev(nullptr), next(nullptr) {}
};
class DoublyLinkedList {
private:
    DoublyNode* head;
    DoublyNode* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}
    void insertBeginning(int data) {
        cout << "Inserting " << data << " at the beginning." << endl;
```

```
    DoublyNode* newNode = new DoublyNode(data);
    if (!head) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void insertEnd(int data) {
    cout << "Inserting " << data << " at the end." << endl;
    DoublyNode* newNode = new DoublyNode(data);
    if (!head) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void deleteBeginning() {
    if (!head) {
        cout << "List is empty. Nothing to delete from the beginning." << endl;
        return;
    }
    cout << "Deleting node from the beginning." << endl;
    DoublyNode* temp = head;
    if (head == tail) {
        head = tail = nullptr;
    } else {
        head = head->next;
        head->prev = nullptr;
    }
    delete temp;
}

void deleteEnd() {
    if (!head) {
        cout << "List is empty. Nothing to delete from the end." << endl;
        return;
```

```
        }
        cout << "Deleting node from the end." << endl;
        DoublyNode* temp = tail;
        if (head == tail) {
            head = tail = nullptr;
        } else {
            tail = tail->prev;
            tail->next = nullptr;
        }
        delete temp;
    }
    void printList() const {
        cout << "Current list: ";
        DoublyNode* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
int main() {
    DoublyLinkedList dcc;
    dcc.insertBeginning(1);
    dcc.insertEnd(2);
    dcc.insertBeginning(0);
    dcc.printList();
    dcc.deleteBeginning();
    dcc.printList();
    dcc.deleteEnd();
    dcc.printList();
    return 0;
}
```

Insertion and Deletion of the element at the beginning and at the end in Circuly Linked List

```
#include <iostream>
using namespace std;
class CircularNode {
public:
    int data;
```

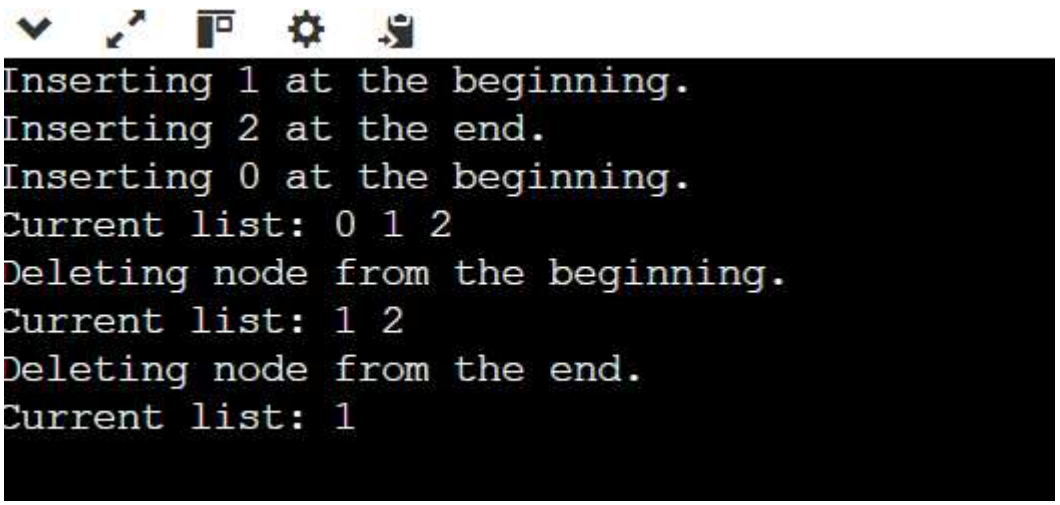
```
    CircularNode* next;

    CircularNode(int data) : data(data), next(nullptr) {}
};

class CircularLinkedList {
private:
    CircularNode* head;

public:
    CircularLinkedList() : head(nullptr) {}
    void insertBeginning(int data) {
        cout << "Inserting " << data << " at the beginning." << endl;
        CircularNode* newNode = new CircularNode(data);
        if (!head) {
            head = newNode;
            newNode->next = head;
        } else {
            newNode->next = head;
            CircularNode* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = newNode;
            head = newNode;
        }
    }

    void insertEnd(int data) {
        cout << "Inserting " << data << " at the end." << endl;
        CircularNode* newNode = new CircularNode(data);
        if (!head) {
            head = newNode;
            newNode->next = head;
        } else {
            CircularNode* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->next = head;
        }
    }
```

```
void deleteBeginning() {
    if (!head) {
        cout << "List is empty. Nothing to delete from the beginning." << endl;
        return;
    }
    cout << "Deleting node from the beginning." << endl;
    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        CircularNode* temp = head;
        CircularNode* current = head;
        while (current->next != head) {
            current = current->next;
        }
        head = head->next;
        current->next = head;
        delete temp;
    }
}
void deleteEnd() {
    if (!head) {
        cout << "List is empty. Nothing to delete from the end." << endl;
        return;
    }
    cout << "Deleting node from the end." << endl;
    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        CircularNode* temp = head;
        CircularNode* prev = nullptr;
        while (temp->next != head) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = head;
        delete temp;
    }
}
void printList() const {
    if (!head) {
```

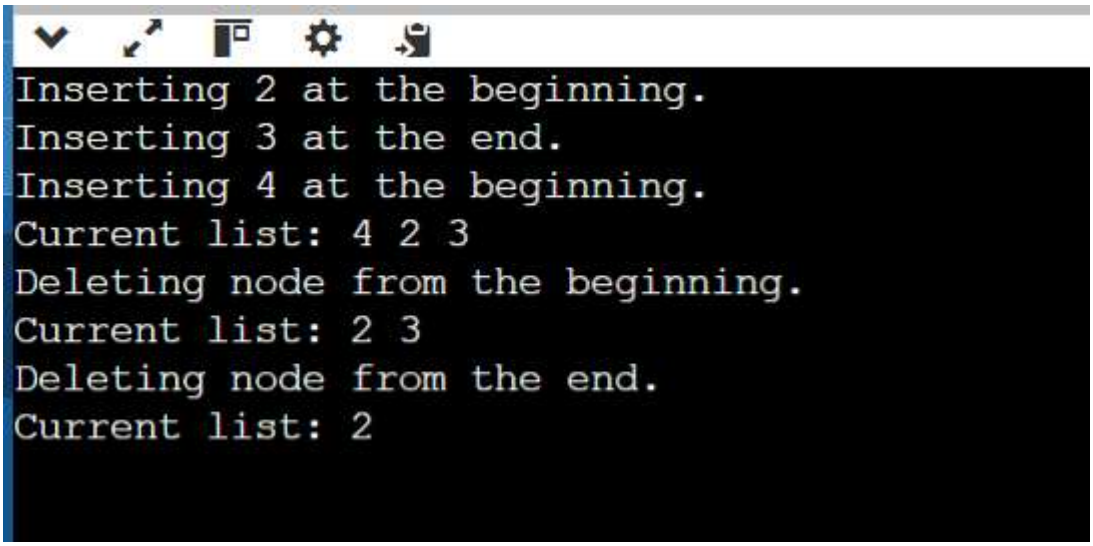
```
            cout << "List is empty." << endl;
            return;
        }
        cout << "Current list: ";
        CircularNode* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << endl;
    }
};
int main() {
    CircularLinkedList cll;
    cll.insertBeginning(2);
    cll.insertEnd(3);
    cll.insertBeginning(4);
    cll.printList();
    cll.deleteBeginning();
    cll.printList();
    cll.deleteEnd();
    cll.printList();
    return 0;
}
```

4.Output:

Doubly Linked List



Circular Linked List



3. **Time Complexity**
   Doubly Linked List

1.Insertion at the Beginning:
- Time Complexity**:** O(1)

2. Insertion at the End:
- Time Complexity**:** O(1)

3. Deletion from the Beginning:
- Time Complexity**:** O(1)

4.Deletion from the End:
- Time Complexity**:** O(1)

Circular Linked List

1.Insertion at the Beginning:
- Time Complexity: O(n)

2.Insertion at the End:
- Time Complexity**:** O(n)

3. Deletion from the Beginning:
- Time Complexity**:** O(n)

4. Deletion from the End:
- Time Complexity**:** O(n)