BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

Munshi nagar, Andheri (W) ,Mumbai - 400058

DEPARTMENT OF MASTER OF COMPUTER APPLICATION

CLASS: F.Y. MCA **SEM: I**

COURSE CODE: MC501 **SUBJECT NAME: DATA STRUCTURES LAB**

ROLL NO. : _2023510001___ **BATCH: _D_**

NAME:__VAIBHAV AGARWAL____

EXPERIMENT NO: 04

EXPERIMENT TITLE: Implement doubly queue using link representation. Read records of books and store in the DQ. Perform insert and delete operations and do traversals in both the orders.

CODE:

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

struct Book {
    string title;
    string author;
    int year;
};

struct Node {
    Book data;
    Node* next;
    Node* prev;
};

class Deque {
private:
    Node* front;
    Node* rear;

public:
    Deque() : front(nullptr), rear(nullptr) {}

    bool isEmpty() {
        return front == nullptr;
```

```
}

void insertFront(const Book& book) {
    Node* newNode = new Node;
    newNode->data = book;
    newNode->next = nullptr;
    newNode->prev = nullptr;

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        newNode->next = front;
        front->prev = newNode;
        front = newNode;
    }
}

void insertRear(const Book& book) {
    Node* newNode = new Node;
    newNode->data = book;
    newNode->next = nullptr;
    newNode->prev = nullptr;

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        newNode->prev = rear;
        rear->next = newNode;
        rear = newNode;
    }
}

void deleteFront() {
    if (isEmpty()) {
        cout << "Deque is empty. Cannot delete from front." << endl;
        return;
    }

    Node* temp = front;
    if (front == rear) {
        front = rear = nullptr;
    } else {
        front = front->next;
        front->prev = nullptr;
    }
    delete temp;
}

void deleteRear() {
    if (isEmpty()) {
        cout << "Deque is empty. Cannot delete from rear." << endl;
        return;
```

```
    }

        Node* temp = rear;
        if (front == rear) {
            front = rear = nullptr;
        } else {
            rear = rear->prev;
            rear->next = nullptr;
        }
        delete temp;
    }

    void traverseFrontToRear() {
        Node* current = front;
        while (current != nullptr) {
            displayBook(current->data);
            current = current->next;
        }
    }

    void traverseRearToFront() {
        Node* current = rear;
        while (current != nullptr) {
            displayBook(current->data);
            current = current->prev;
        }
    }

    void displayBook(const Book& book) {
        cout << "Title: " << book.title << ", Author: " << book.author << ", Year: " << book.year << endl;
    }
};

int main() {
    Deque bookDeque;

    Book book1 = { "Book1", "Author1", 2020 };
    Book book2 = { "Book2", "Author2", 2019 };
    Book book3 = { "Book3", "Author3", 2018 };

    bookDeque.insertFront(book1);
    bookDeque.insertRear(book2);
    bookDeque.insertRear(book3);

    cout << "Deque Contents (Front to Rear):" << endl;
    bookDeque.traverseFrontToRear();

    cout << "\nDeque Contents (Rear to Front):" << endl;
    bookDeque.traverseRearToFront();

    return 0;
}
```

OUTPUT:

```
Deque Contents (Front to Rear):
Title: Book1, Author: Author1, Year: 2020
Title: Book2, Author: Author2, Year: 2019
Title: Book3, Author: Author3, Year: 2018

Deque Contents (Rear to Front):
Title: Book3, Author: Author3, Year: 2018
Title: Book2, Author: Author2, Year: 2019
Title: Book1, Author: Author1, Year: 2020
```