Doubly Linked List Implementation in C++

Objective:
The objective of this task is to understand and implement a Doubly Linked List using C++. This task demonstrates dynamic memory allocation, pointer manipulation, and traversal in both forward and backward directions.

Problem Statement

Implement a Doubly Linked List with the following operations:
- Insert nodes at the end of the list
- Display the list in forward direction
- Display the list in backward direction

Concepts Used

- Doubly Linked List
- Dynamic Memory Allocation
- Pointers
- Classes and Objects
- Traversal Techniques

Source Code (C++)

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

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

        Node(int value) {
        data = value;
        prev = NULL;
```

```
        next = NULL;
        }
};

class DoublyLinkedList {
private:
        Node* head;

public:
        DoublyLinkedList() {
        head = NULL;
        }

        void insertEnd(int value) {
        Node* newNode = new Node(value);

        if (head == NULL) {
        head = newNode;
        return;
        }

        Node* temp = head;
          while (temp->next != NULL) {
        temp = temp->next;
        }

        temp->next = newNode;
        newNode->prev = temp;
        }

        void displayForward() {
        Node* temp = head;
        cout << "Forward Traversal: ";
        while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
        }
        cout << endl;
        }

        void displayBackward() {
        Node* temp = head;
        if (temp == NULL) return;

        while (temp->next != NULL) {
          temp = temp->next;
        }
```

```
        cout << "Backward Traversal: ";
        while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->prev;
        }
        cout << endl;
        }
};

int main() {
        DoublyLinkedList dll;

        dll.insertEnd(10);
        dll.insertEnd(20);
        dll.insertEnd(30);
        dll.insertEnd(40);

        dll.displayForward();
        dll.displayBackward();

        return 0;
}
```

Explanation of the Code

1. The Node class represents each node containing data, a pointer to the previous node, and a pointer to the next node.
2. The DoublyLinkedList class maintains the head pointer.
3. insertEnd() inserts a new node at the end of the list.
4. displayForward() traverses the list from head to tail.
5. displayBackward() traverses the list from tail to head.
6. Memory is allocated dynamically using the new keyword.
7. The main() function demonstrates insertion and traversal operations.

Program Output

```
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$ g++ doubly_linked_list.cpp
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$ ./a.out
Forward Traversal: 10 20 30 40
Backward Traversal: 40 30 20 10
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$
```

Conclusion

This task demonstrates the implementation of a Doubly Linked List in C++. Each node maintains links to both previous and next nodes, allowing bidirectional traversal. Doubly linked lists are commonly used in navigation systems, undo-redo operations, and memory management.