THE UNIVERSITY OF AZAD JAMMU AND KASHMIR, MUZAFFARABAD

Department of Software Engineering

Submitted to:	Engr. Sidra Rafique
Course Title:	Data Structure and Algorithm
Course Code:	CS-2101
Session:	2024-2028
Lab No:	04
Roll No:	2024-SE-15
Submitted By:	Shahzad Ahmed Awan
Submission date:	November 1, 2025

Table of Contents

Table of Figures

Lab 04 – Doubly Linked List Implementation

1. Objective

To implement a Doubly Linked List (DLL) in C++ that performs insertion and deletion at both ends and displays data in forward and backward directions using a simple menu-driven interface.

2. Background

A Doubly Linked List consists of nodes linked in both directions using prev and next pointers.
It allows easy traversal and modification from either end, making operations efficient compared to a singly linked list.

3. Algorithm

1. Start the program.
2. Define a Node structure with data, prev, and next pointers.
3. Create a DoublyLinkedList class with head and tail pointers.
4. Implement:
    - insertAtBeginning() – adds a node at the start.
    - insertAtEnd() – adds a node at the end.
    - deleteFromBeginning() – removes the first node.
    - deleteFromEnd() – removes the last node.
    - displayForward() – shows list from head to tail.
    - displayBackward() – shows list from tail to head.
5. In main(), use a loop-based menu for repeated operations.
6. Clear and update the screen after each operation for clarity.
7. Stop on exit.

---

4. Source Code Explanation

4.1 Insert at Beginning

```
// Insert node at the beginning
void insertAtBeginning(int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
    cout << "\nInserted " << value << " at the beginning.\n";
}
```

Figure 1: Code for inserting a node at the beginning of the doubly linked list.

Creates a new node and links it before the current head.
If the list is empty, both head and tail point to the new node.

4.2 Insert at End

Adds a new node after the tail.
Updates the previous and next pointers to maintain proper linkage.

```
// Insert node at the end
void insertAtEnd(int value) {
    Node* newNode = new Node(value);
    if (!tail) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
    cout << "\nInserted " << value << " at the end.\n";
}
```

Figure 2: Code for inserting a node at the end of the doubly linked list.

4.3 Delete from Beginning

Removes the first node and reassigns head.
If only one node exists, both pointers become NULL.

```
// Delete node from the beginning
void deleteFromBeginning() {
    if (!head) {
        cout << "\nList is empty. Nothing to delete.\n";
        return;
    }
    Node* temp = head;
    if (head == tail) { // Only one node
        head = tail = nullptr;
    } else {
        head = head->next;
        head->prev = nullptr;
    }
    cout << "\nDeleted " << temp->data << " from the beginning.\n";
    delete temp;
}
```

Figure 3: Code for deleting a node from the beginning of the list.

4.4 Delete from End

Deletes the last node and updates the tail pointer.
Handles empty-list conditions safely.

```cpp
// Delete node from the end
void deleteFromEnd() {
    if (!tail) {
        cout << "\nList is empty. Nothing to delete.\n";
        return;
    }
    Node* temp = tail;
    if (head == tail) { // Only one node
        head = tail = nullptr;
    } else {
        tail = tail->prev;
        tail->next = nullptr;
    }
    cout << "\nDeleted " << temp->data << " from the end.\n";
    delete temp;
}
```

Figure 4: Code for deleting a node from the end of the list.

4.5 Display Forward and Backward

Forward display prints nodes like 9 -> 6 -> 3 -> NULL.
Backward display shows them in reverse order using prev pointers.

```cpp
// Display list in forward direction
void displayForward() {
    if (!head) {
        cout << "\nList is empty.\n";
        return;
    }
    cout << "\nList (Forward): ";
    Node* temp = head;
    while (temp) {
        cout << temp->data;
        if (temp->next) cout << " -> ";
        temp = temp->next;
    }
    cout << " -> NULL\n";
}
```

Figure 6: Code for displaying the list in the forward directions.

```cpp
// Display list in backward direction
void displayBackward() {
    if (!tail) {
        cout << "\nList is empty.\n";
        return;
    }
    cout << "\nList (Backward): ";
    Node* temp = tail;
    while (temp) {
        cout << temp->data;
        if (temp->prev) cout << " -> ";
        temp = temp->prev;
    }
    cout << " -> NULL\n";
}
```

Figure 5: Code for displaying the list in backward directions.

4.6 Menu Loop

The main program runs inside a loop allowing multiple operations.
After each action, the screen refreshes, giving a clean and user-friendly display.

```
// Main Function
int main() {
    DoublyLinkedList dll;
    int choice, value;

    do {
        cout << "========================================\n";
        cout << "           DOUBLY LINKED LIST MENU          \n";
        cout << "========================================\n";
        cout << "1. Insert at Beginning\n";
        cout << "2. Insert at End\n";
        cout << "3. Delete from Beginning\n";
        cout << "4. Delete from End\n";
        cout << "5. Display Forward\n";
        cout << "6. Display Backward\n";
        cout << "0. Exit\n";
        cout << "----------------------------------------\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cout << "----------------------------------------\n";

        switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            dll.insertAtBeginning(value);
            pauseAndClear();
            break;
        case 2:
            cout << "Enter value to insert: ";
            cin >> value;
            dll.insertAtEnd(value);
            pauseAndClear();z
            break;
        case 3:
            dll.deleteFromBeginning();
            pauseAndClear();
            break;
        case 4:
            dll.deleteFromEnd();
            pauseAndClear();
            break;
        case 5:
            dll.displayForward();
            pauseAndClear();
            break;
        case 6:
            dll.displayBackward();
            pauseAndClear();
            break;
        case 0:
            cout << "\nExiting Program. Goodbye!\n";
            break;
        default:
            cout << "\nInvalid choice! Try again.\n";
            pauseAndClear();
        }

    } while (choice != 0);

    return 0;
}
```

Figure 7: Stimulation using main function

5. Output

The program provides a menu where users can insert, delete, or display nodes.
Each operation produces clear, bordered output and updates dynamically after every action.

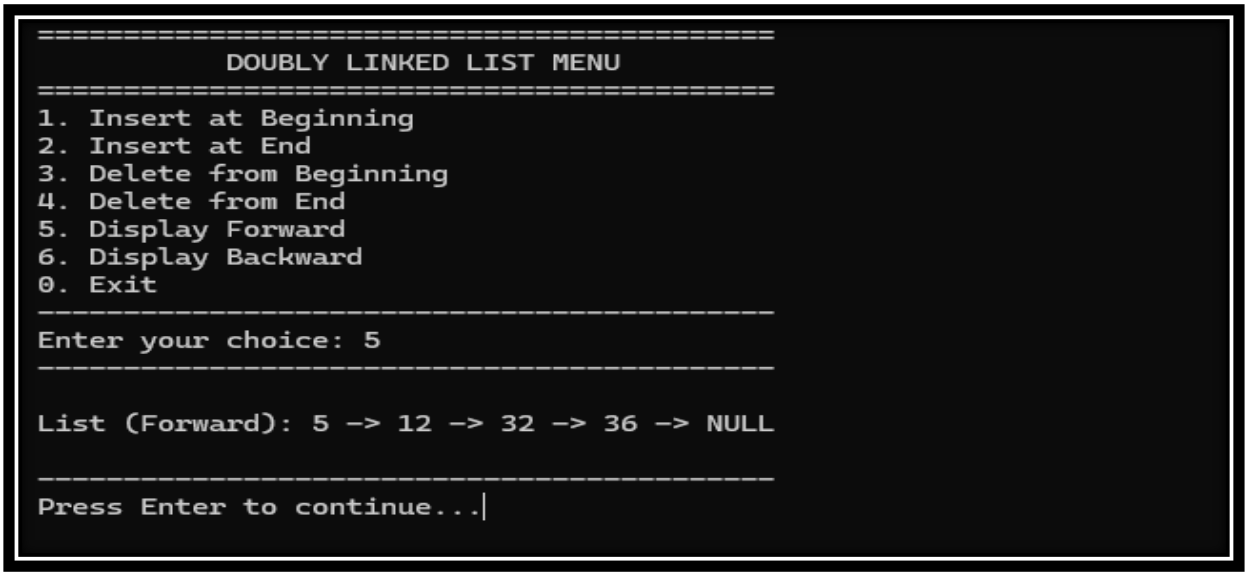

Figure 9: Output showing list forward

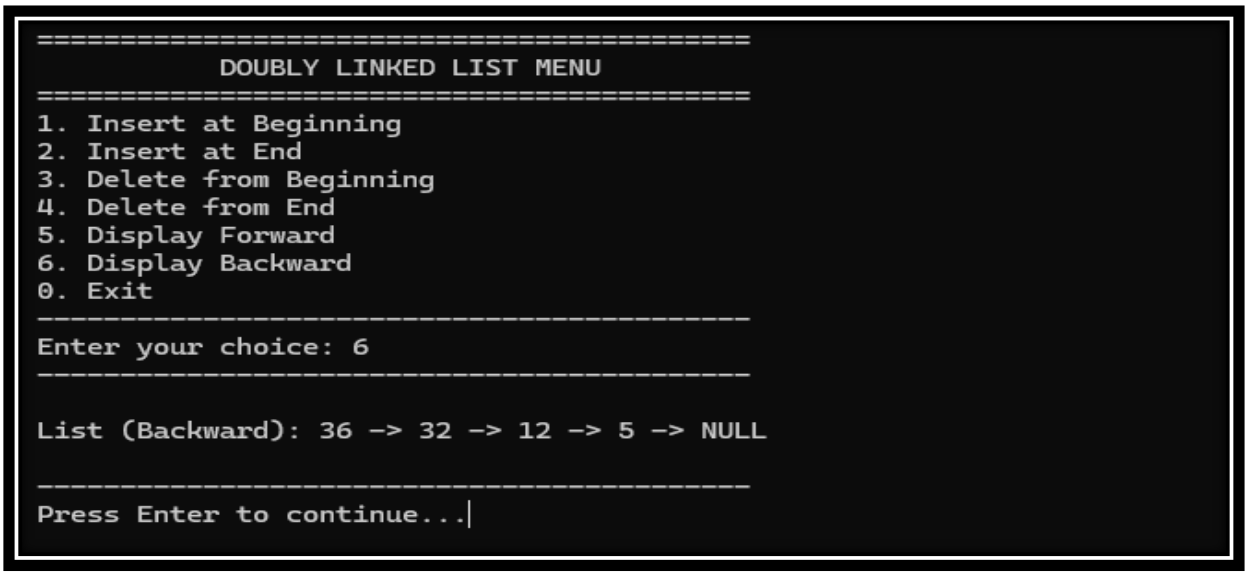

Figure 8: Output Showing list backward

6. Conclusion

The lab successfully implements a Doubly Linked List with insertion, deletion, and bidirectional traversal.
It demonstrates pointer manipulation and dynamic data handling in C++ effectively.

---

7. Reflection

This lab improved understanding of linked list operations, pointer logic, and loop-based interaction in C++.
It also showed how simple formatting enhances program readability and usability.