AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

SE(ECE)	**Data Structures and Algorithms**	
Experiment No: 5	**Implement Doubly Linked List with insertion, deletion, and display operations**	**Page: 1/4**

Aim

To implement a Doubly Linked List (DLL) with the following operations:

- Insertion of nodes at the end.
- Deletion of a node by value.
- Displaying the list in forward and backward directions.

Objectives

1. To understand the concept of a Doubly Linked List**.**
2. To implement insertion and deletion operations using pointers.
3. To perform traversal of a linked list.

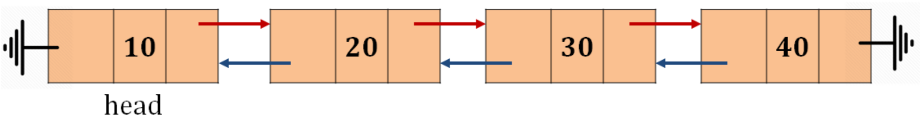
Theory

Doubly Linked List (DLL)

A Doubly Linked List is a linear data structure consisting of nodes. Each node has three parts:

1. **Data field** – stores the element.
2. **Pointer to the next node (next)** – points to the next node.
3. **Pointer to the previous node (prev)** – points to the previous node.

This allows traversal in both forward and backward directions.



A doubly linked list

Node Structure

```
class node {
public:
   int data;
   node* prev;
   node* next;
};
```

Algorithms

Insertion at the End

1. Create a new node p.
2. If list is empty (head == NULL), set head = p.
3. Otherwise, traverse list using q until last node.
4. Attach new node:
   - q->next = p
   - p->prev = q

Deletion by Value

1. Use pointer q to search for the node.
2. If head node matches:
   - Assign p = head.
   - Move head = head->next.
   - Set head->prev = NULL.
   - Delete p.
3. Otherwise, traverse until node found:
   - p = q->next
   - q->next = p->next
   - p->next->prev = q
   - Delete p.
4. If node not found, display message.

Display Forward

1. Start from head node q.
2. Traverse until q == NULL, printing q->data.

Display Backward

1. Start from head node q and move to last node.
2. Traverse backward (q = q->prev) until start of list, printing q->data.

Applications of Doubly Linked List

- Undo/Redo functionality in text editors.
- Back/Forward navigation in web browsers.
- Implementation of Deque, Polynomial Manipulation.
- Memory management in operating systems.

References

1. Yedidyah Langsam, Moshe J Augenstein, Aaron M Tenenbaum – Data structures using C and C++ - PHI Publications (2nd Edition).
2. Ellis Horowitz, Sataraj Sahni- Fundamentals of Data Structures – Galgotia Books source.

Questions

1. In a doubly linked list, each node contains:
a) Data and one pointer
b) Data and two pointers
c) Only data
d) Only one pointer

2. Which of the following is an advantage of a doubly linked list over a singly linked list?
a) Requires less memory per node
b) Traversal is possible in both directions
c) Insertion and deletion at the end are faster
d) Implementation is easier

3. What does the `prev` pointer in a doubly linked list node store?
a) The address of the next node
b) The address of the previous node
c) The address of the first node
d) The address of the last node

4. Time complexity to insert a node at the end of a doubly linked list is:
a) O(1) if tail pointer is maintained
b) O(n) if tail pointer is not maintained
c) $O(n^2)$
d) Both a and b depending on implementation

5. Deleting a node with a given value from a doubly linked list requires:
a) Only updating the `next` pointer of the previous node
b) Only updating the `prev` pointer of the next node
c) Updating both `next` and `prev` pointers of adjacent nodes
d) No pointer updates are needed

6. If a doubly linked list has n nodes, how many NULL links will it contain?
a) 0
b) 1
c) 2
d) n

7. Which traversal is possible in a doubly linked list but not in a singly linked list?
a) Forward traversal
b) Backward traversal
c) Random access traversal
d) Circular traversal

8. What happens when you delete the head node of a doubly linked list?
a) Only `prev` pointer is updated
b) Only `next` pointer is updated
c) The new head's `prev` pointer is set to NULL
d) The list cannot delete the head node

9. Which of the following applications can use a doubly linked list?
a) Implementing Undo/Redo functionality in editors
b) Navigation in web browsers (forward/backward)
c) Deques (double-ended queues)
d) All of the above

10. Compared to arrays, doubly linked lists:
a) Provide constant time random access
b) Are more memory efficient
c) Have faster insertions and deletions (if node reference is known)
d) Are easier to implement

Conclusion

A	C	O	T	Sign.