

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering
Computer Engineering Department
Program: B.Tech Integrated. Sem V

Course: Basic Data Structures
LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.8

A.1 Aim:

To study and implement the concept of the doubly linked list data structure.

A.2 Prerequisite:

- Understanding of the basic operations on linked list data structures.
- Knowledge of doubly linked lists and their applications.
- Familiarity with C/C++ programming.

A.3 Outcome:

After completing this experiment, students will be able to:

- Identify the need for a doubly linked list in specific scenarios.
- Implement a doubly linked list.
- Perform insertion, deletion, and traversal operations on a doubly linked list.
- Understand the difference between singly and doubly linked lists.

A.4 Theory:

A.4.1 Introduction to Doubly Linked List:

A doubly linked list is a linear data structure where each node contains three fields:

1. **Data:** The actual value.
2. **Next:** A pointer to the next node in the list.
3. **Previous:** A pointer to the previous node in the list.

The presence of a "previous" pointer makes it possible to traverse the list in both forward and backward directions, unlike singly linked lists. This provides greater flexibility but requires additional memory to store the previous pointers.

Advantages of Doubly Linked Lists:

- **Bidirectional traversal:** You can traverse the list in both directions.
- **Easier node deletion:** Access to both previous and next nodes simplifies node deletion.
- **Efficient operations:** Insertion and deletion can be more efficient, especially when a reference to the node is available.

Disadvantages:

- **Increased memory usage:** Each node requires extra memory to store the previous pointer.
- **More complex implementation:** Maintaining both pointers adds complexity to operations like insertion, deletion, and traversal.

A.5 Procedure/Algorithm:

Insertion in a Doubly Linked List:

1. Create a new node.
2. Set the new node's next pointer to the current node's next.
3. Set the new node's previous pointer to the current node.
4. Adjust the previous and next pointers of adjacent nodes to insert the new node into the list.

Deletion from a Doubly Linked List:

1. Update the previous pointer of the next node (if any) to point to the node's previous node.
2. Update the next pointer of the previous node (if any) to point to the node's next node.
3. Free the memory occupied by the node being deleted.

(Task 1):

Write a C/C++ program to implement the following operations on a doubly linked list:

- Insert a node at the beginning
- Insert a node at the end
- Delete a node at the beginning
- Delete a node at the end

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll No.	Name:
Class :	Batch :
Date of Experiment:	Date of Submission
Grade :	Time of Submission:
Date of Grading:	

B.1 Software Code written by student: (Task 1)

(Paste your code completed during the 2 hours of practical in the lab here)

Task1:

```
#include<iostream>

#include<stdlib.h>

using namespace std;

struct node {

    struct node *prev;

    int data;

    struct node *next;

};

struct node *head ,*tail ,*temp ,*newnode;

int c=0;
```

```
void createnewnode(int x)
{
    newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=x;
    newnode->prev=0;
    newnode->next=0;
    if(head==NULL)
    {
        head=newnode;
        tail=newnode;
    }
    else
    {
        tail->next=newnode;
        newnode->prev=tail;
        tail=newnode;
    }
    c++;
}
```

```
void insertatbeg(int x)
{
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=x;
```

```

newnode->prev=0;
newnode->next=0;

head->prev=newnode; // 100 0 3 300 300 1 200
newnode->next=head;
newnode->prev=0;
head=newnode;
c++;
}
void inseratend(int x)
{
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=x;
    newnode->next=0;
    newnode->prev=0;

    newnode->prev=tail;
    tail->next=newnode;
    tail=newnode;
    c++;
}
void display()
{

```

```
if(c==0)
{
    cout<<"\n List Is Empty !! ";
}
else
{
    cout<<"\n List : ";
    temp=head;
    while(temp!=0)
    {
        cout<<temp->data<<" ";
        temp=temp->next;
    }
}

void displayrev()
{
    if(c==0)
    {
        cout<<"\n List Is Empty !! ";
    }
    else
    {
```

```

        cout<<"\n List : ";

        temp=tail;

        while(temp!=0)

        {

            cout<<temp->data<<" ";

            temp=temp->prev;

        }

    }

}

void deleteatbeg()

{

    temp=head;

    head=head->next;

    head->prev=0;

    free(temp);

    c--;

}

void deleteatend()

{

    temp=tail;

    tail=tail->prev;

    tail->next=0;

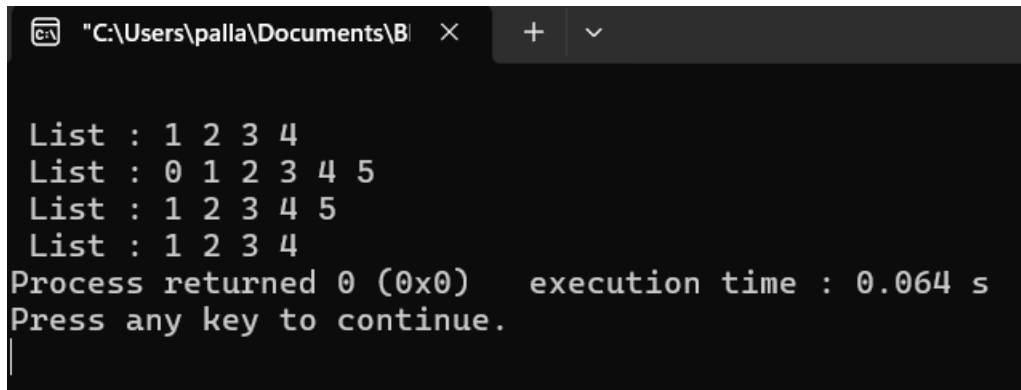
    free(temp);

```

```
    c--;  
}  
int main()  
{  
    createnewnode(1);  
    createnewnode(2);  
    createnewnode(3);  
    createnewnode(4);  
    display();  
    insertatbeg(0);  
    inseratend(5);  
    display();  
    deleteatbeg();  
    display();  
    deleteatend();  
    display();  
}
```


B.2 Input and Output: (Task 1)

(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)



```
"C:\Users\palla\Documents\B"
List : 1 2 3 4
List : 0 1 2 3 4 5
List : 1 2 3 4 5
List : 1 2 3 4
Process returned 0 (0x0)   execution time : 0.064 s
Press any key to continue.
```

Task1:

B.3 Observations and learning [w.r.t. all tasks]:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)

Topics Understood :-

1. creation of doubly linked list node.
2. Insertion at the start of doubly linked list.
3. Insertion at the end of doubly linked list.
4. Deletion at the start of doubly linked list.
5. Deletion at the end of doubly linked list.
6. Displaying Doubly Linked list in both directions using prev and next pointer.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Learnt how to implement Doubly Linked List ADT.

B.5 Question of Curiosity

(To be answered by student based on the practical performed and learning/observations)

State the advantages of using a doubly linked list over a singly linked list.

- **Two-way traversal:** In a doubly linked list, you can traverse the list in both directions (forward and backward), whereas in a singly linked list, you can only traverse in one direction.
- **Efficient deletion:** Deleting a node in a doubly linked list is more efficient since you can directly access the previous node (using the backward pointer), while in a singly linked list, you need to traverse from the head to find the node before the one to be deleted.
- **Easier insertion at the end:** Inserting a node at the end of a doubly linked list can be done efficiently if you maintain a pointer to the last node, as you can access it directly. In a singly linked list, you would need to traverse the entire list to reach the last node.
- **Reversing the list:** Reversing a doubly linked list is simpler because you have pointers to both the next and previous nodes, whereas reversing a singly linked list requires more complex manipulation and extra memory.