Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
COMPS Department

Experiment	4
Aim	**perform following operations on doubly linked lists 1. Create 2. find the length of list using recursion 3. delete a node 4. display**
Name	David Daniels
UID	202330038
Class	Div -A
Batch	C
Date of Submission	11-9-24

Theory	A **doubly linked list** is a data structure that consists of a set of nodes, each of which contains a **value** and **two pointers**, one pointing to the **previous node** in the list and one pointing to the **next node** in the list. This allows for efficient traversal of the list in **both directions**, making it suitable for applications where frequent **insertions** and **deletions** are required.
	
	In a data structure, a doubly linked list is represented using nodes that have three fields:
	1. Data
	2. A pointer to the next node (**next**)
	3. A pointer to the previous node (**prev**)
	
	We wrap the data item and the next,previous node reference in a struct as:
	struct Node {
	int data;
	Node* prev;
	Node* next;
	};
	
	A Doubly linked list is an ADT as we can implement many operations in it to insert elements at any location or delete at any location or simply traverse the list or find its length.
	
	1. Insertion: This to add an element in the Doubly linked list. The implementation of this function will be different depending on the location. Insertion can be done on first , last and middle node
	
	2. Deletion: This to remove an element in the Doubly linked list. The implementation of this function will be different depending on the location. Deletion can be done on first , last and middle node

3. Find Length: This to find the length of list or the number of nodes in it. It can be done by using traversal method or by using recursion.

4. Traversal: This is to pass through all elements of list , doing some operation on all of them

Difference between Singly linked list and Doubly linked list

1. Singly linked list

- SLL nodes contains 2 field -data field and next link field.
- In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.
- Supports lesser number of operations in constant time
- The SLL occupies less memory than DLL as it has only 2 fields.
- Singly linked list is relatively less used in practice due to limited number of operations

2. Doubly Linked list

- DLL nodes contains 3 fields -data field, a previous link field and a next link field.
- In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).
- Supports additional operations like insert before, delete previous, delete current node and delete last in constant time
- The DLL occupies more memory than SLL as it has 3 fields.
- The doubly linked list consumes more memory as compared to the singly linked list.
- Doubly linked list is implemented more in libraries due to wider number of operations.

Advantages Of DLL:

- Reversing the doubly linked list is very easy.
- It can allocate or reallocate memory easily during its execution.
- The traversal of this doubly linked list is bidirectional which is not possible in a singly linked list.
- Deletion of nodes is easy as compared to a Singly Linked List. A singly linked list deletion requires a pointer to the node and previous node to be deleted but in the doubly linked list, it only required the pointer which is to be deleted.'

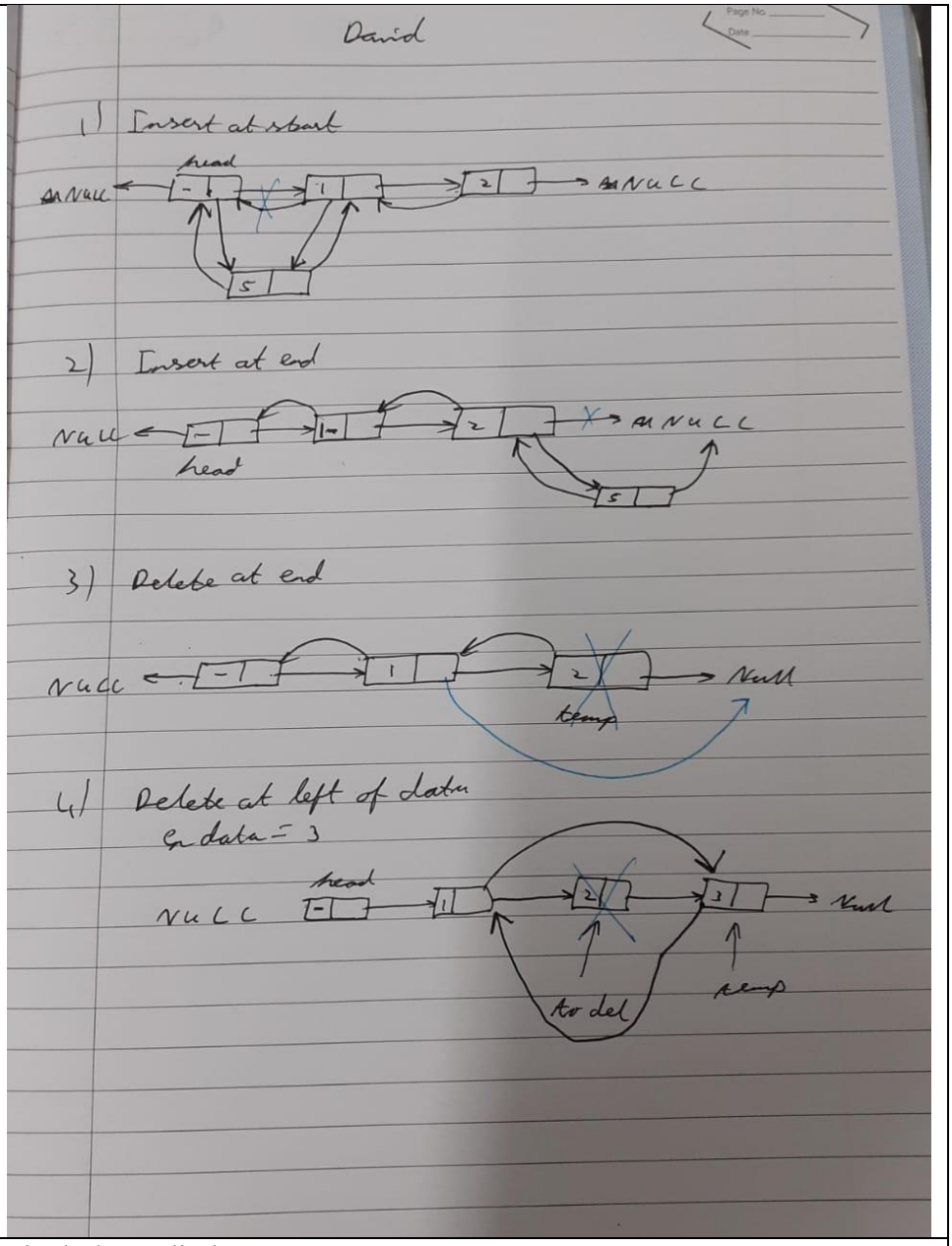
Disadvantages Of DLL:

- It uses extra memory when compared to the array and singly linked list because it requires an extra pointer in every node.
- Traversing a doubly linked list can be slower than traversing a singly linked list.
- Implementing and maintaining doubly linked lists can be more complex than singly linked lists.

	Applications Of DLL: • It is used in the navigation systems where front and back navigation is required. • It is used by the browser to implement backward and forward navigation of visited web pages that is a back and forward button. • It is also used to represent a classic game deck of cards. • It is also used by various applications to implement undo and redo functionality. • Doubly Linked List is also used in constructing **MRU/LRU** (Most/least recently used) cache. • Other data structures like stacks, Hash Tables, Binary trees can also be constructed or programmed using a doubly-linked list. • Also in many operating systems, the **thread scheduler**(the thing that chooses what process needs to run at which time) maintains a doubly-linked list of all processes running at that time. • Implementing LRU Cache. • Implementing Graph algorithms.
Algorithm	1. Insertion 1) At the Beginning • Create a new node, say **new_node** with its previous pointer as NULL. • Set the next pointer to the current head, **new_node->next = head**. • Check if the linked list is not empty then we update the previous pointer of the current head to the new node, **head->prev = new_node**. • Finally, we return the new node as the head of the linked list. 2) At the End • Allocate memory for a new node, say **new_node** and assign the provided value to its data field. • Initialize the next pointer of the new node to NULL, **new_node->next = NULL**. • If the linked list is empty, we set the new node as the head of linked list and return it as the new head of the linked list. • Traverse the entire list until we reach the last node, say **curr**. • Set the next pointer of last node to new node, **curr->next = new_node** • Set the prev pointer of new node to last node, **new_node->prev = curr** 3) At a given index To insert a new node at a specific position, • Traverse the list to position – 1.

	• If the position is valid, update the next pointer of **new node** to the next of **current node** and prev pointer of new node to the current node. • Similarly, update next pointer of current node to the **new node** and prev pointer of new node's next to the new node**.** 2. Deletion 1) At the Beginning • Check if the **list** is empty, there is nothing to **delete**, return. • Store the **head** pointer in a variable, say **temp**. • Update the **head** of linked list to the node **next** to the **current head**, **head = head->next**. • If the new head is not NULL, update the **previous pointer** of new head to NULL, **head->prev = NULL**. 2) At the End • Check if the doubly linked list is empty. If it is empty, then there is nothing to delete. • If the list is not empty, then move to the **last node** of the doubly linked list, say **curr**. • Update the **second-to-last** node's next pointer to **NULL**, **curr->prev->next = NULL**. • Free the memory allocated for the node that was deleted. 3) To the left of a node (in the middle) • Initialize a variable , say **curr** points to the node with the **key** value in the linked list. • if found , check if **curr->p**rev is not **NULL**. o If it's **NULL**, the node to be deleted is the **head node**, so there is no node to delete before it. o else , set a pointer **nodeDelete** to **curr->prev**, which is the node to be **deleted**. o Update **curr->prev** to point to **nodeDelete ->prev.** o If **nodeDelete ->prev** is not **NULL**, update **nodeDelete->prev->next** point to **curr**. • Delete **nodeDelete** to free memory. Similar Logic can be used to delete at right of node 4) Deletion at specific node □ raverse to the node at the specified position, say **curr**. □ If the position is valid, adjust the pointers to skip the node to be deleted. • If curr is not the head of the linked list, update the next pointer of the node before curr to point to the node after curr, **curr->prev->next = curr->next**. • If curr is not the last node of the linked list, update the **previous pointer** of the node after curr to the node before curr, **curr->next->prev = curr->prev**.

	Free the memory allocated for the deleted node. 3. Length (using recursion) • **Base Case**: This condition determines whether the current node (head) is NULL, which signifies the end of the linked list or that the list is empty. If head is NULL, the function returns 0, indicating that there are no more nodes to count. • **Recursive Case**: When head is not NULL, the function proceeds to count the current node. return 1 + countNodes(head->next) accomplishes this 4. Display 1) Initialize a pointer to the head of the linked list. 2) While the pointer is not null: 3) Visit the data at the current node. 4) Move the pointer to the next node.

Problem Solving	

Program(Code)	#include <stdio.h>
	#include <stdlib.h>
	#include <stdbool.h>
	
	typedef struct node
	{
	int data;
	struct node *next;
	struct node * prev;
	}node;
	
	void display (node * head)
	{
	node * temp = head;
	printf("\n");
	while (temp->next!=NULL)

```
    {
        temp=temp->next;
        printf("%d ->", temp->data);
    }
    printf(" NULL \n");
    printf("\n\n");
}

int length (node * head)
{
    if (head==NULL)
    {
        return 0;
    }
    return (1+length(head->next));
}

node * get_node(int data)
{
    node * new = (node * ) malloc(sizeof(node));
    new->data=data;
    new->next=NULL;
    new->prev=NULL;
    return new;
}

node * insert_at_start (node * head , int data)
{
    node * new = get_node(data);
    head->next->prev=new;
    new->next=head->next;
    head->next=new;
    display(head);
    return head;
}

node * insert_at_end (node * head , int data)
{
    node * new = get_node(data);
    node * temp=head;

    while (temp->next!=NULL)
    {
        temp=temp->next;
    }
    temp->next=new;
    new->prev=temp;
    display(head);
    return head;
}
```

```
node * delete_at_end (node * head )
{
   node * temp=head;

   while (temp->next!=NULL)
   {
      temp=temp->next;
   }
   //temp is now last node;
   temp->prev->next=NULL;
   free(temp);
   display(head);
   return head;
}

node * delete_at_left_of_data(node * head , int value)
{
   node * temp=head;
   while (temp->next!=NULL && temp->data!=value)
   {
      temp=temp->next;
   }
   node * to_del = temp->prev;
   //temp is now the node
   // delete the node to the left of it
   temp->prev->prev->next=temp;
   temp->prev = temp->prev->prev;
   free(to_del);
   display(head);
   return head;

}

void menu (node * head)
{
   int choice=0;
   printf("Welcome to choice \n Enter 1 to print \n Enter 2 to insert at
start \n Enter 3 to insert at end \n Enter 4 to delete at end \n Enter 5 to
delete at left of data \n Enter 6 to print length of Doubly LL \n Enter 0 to
end\n");
   int num=0,len=0;
   while (true)
   {
      printf("Enter choice \n");
      scanf("%d",&choice);
      if (choice==0)
      {
         printf("Goodbye");
         break;
```

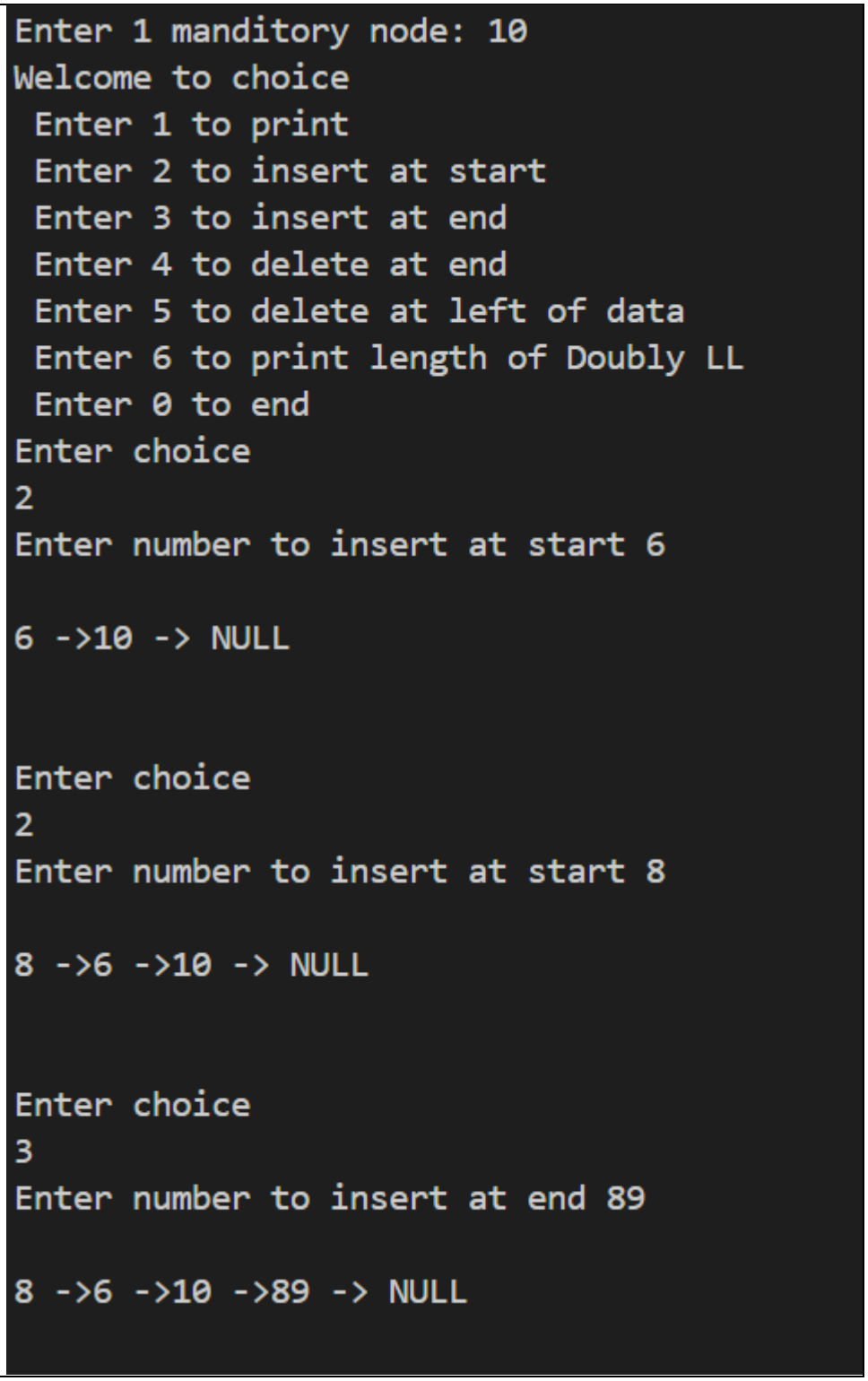
```
        }

        switch (choice)
        {
        case 1:
           display(head);
           break;
        case 2:
           printf("Enter number to insert at start ");
           scanf("%d",&num);
           insert_at_start(head,num);
           break;
        case 3:
           printf("Enter number to insert at end ");
           scanf("%d",&num);
           insert_at_end(head,num);
           break;
        case 4:
           delete_at_end(head);
           break;
        case 5:
           printf("Enter number to delete at left of ");
           scanf("%d",&num);
           delete_at_left_of_data(head,num);
           break;
        case 6:
           len = length(head)-1;
           printf("Length is %d \n\n", len);
           break;
        default:
           printf("Error: ");
           break;
        }
     }
}

int main(int argc, char const *argv[])
{
   node * head = (node * ) malloc(sizeof(node));

   int num=0;
   printf("Enter 1 manditory node: ");
   scanf("%d", &num);
   node * n1 = get_node(num);
   head->next=n1;

   menu(head);
   return 0;
}
```

Output	 Enter 1 manditory node: 10 Welcome to choice Enter 1 to print Enter 2 to insert at start Enter 3 to insert at end Enter 4 to delete at end Enter 5 to delete at left of data Enter 6 to print length of Doubly LL Enter 0 to end Enter choice 2 Enter number to insert at start 6 6 ->10 -> NULL Enter choice 2 Enter number to insert at start 8 8 ->6 ->10 -> NULL Enter choice 3 Enter number to insert at end 89 8 ->6 ->10 ->89 -> NULL

```
Enter choice
3
Enter number to insert at end 25

8 ->6 ->10 ->89 ->25 -> NULL


Enter choice
4

8 ->6 ->10 ->89 -> NULL


Enter choice
5
Enter number to delete at left of 10

8 ->10 ->89 -> NULL


Enter choice
6
Length is 3

Enter choice
0
Goodbye
```

Conclusion	Thus I have learned how to implement doubly linked lists in C and how to perform various insertions and deletions in the same. I have also learned how to use recursion in a linked list.