Syeda Reeha Quasar

14114902719

3C7

Experiment - 6

Data Structures

Aim

Create a Doubly linked list and perform following operations: Insertion at front and Deletion at end.

# **EXPERIMENT – 6**

**AIM:** Create a Doubly linked list and perform following operations: Insertion at front and Deletion at end

# **THEORY**

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2014/03/DLL1.png)

**Advantages over singly linked list**

1. A DLL can be traversed in both forward and backward direction.
2. The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node.
4. In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed.
5. In DLL, we can get the previous node using previous pointer.

**Disadvantages over singly linked list**

1. Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though (See [this](https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-1/)and [this](https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-2/)).
2. All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

**INSERTION IN THE BEGINNING**

## **Source code:**

// insertion in the beginning doubly linked list

//required libraries

#include <stdio.h>

#include <stdlib.h>

// doubly linked list declaration

struct Node {

int data;

struct Node\* next; // Pointer to next node

struct Node\* prev; // Pointer to previous node

};

//function for insertion at the beginning

void insertAtBeginning(struct Node\*\* head\_ref, int newHeadData)

{

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node)); // new node

// assigning value to the new node and assigning values to its pointers

new\_node->data = newHeadData; //storing given data into new node

new\_node->next = (\*head\_ref); //dll next of node to head

new\_node->prev = NULL; // making new node head by pointing previous node to null

if ((\*head\_ref) != NULL) // checking if head is present or not

(\*head\_ref)->prev = new\_node; // changing poiter of headnode from null to new node

(\*head\_ref) = new\_node; // changing head to new node

}

// printing the DLL

void printList(struct Node\* node)

{

struct Node\* last; // declaring a new node for reverse traversal

// traversal in forward direction

printf("\nTraversing in forward direction \n");

while (node != NULL) {

printf(" %d ", node->data);

last = node;

node = node->next;

}

//traversal in reverse direction

printf("\nTraversal in reverse direction \n");

while (last != NULL) {

printf(" %d ", last->data);

last = last->prev;

}

}

int main()

{

// my info

printf("\n\n Name - Syeda Reeha Quasar \n Roll No. - 14114802719 \n Group - 3C7 \n\n");

struct Node\* head = NULL; // declaring head as null

// all these elements are inserted in the beginning

insertAtBeginning(&head, 1); //inserting 1

insertAtBeginning(&head, 2); //inserting 2

insertAtBeginning(&head, 3); //inserting 3

insertAtBeginning(&head, 4); //inserting 4

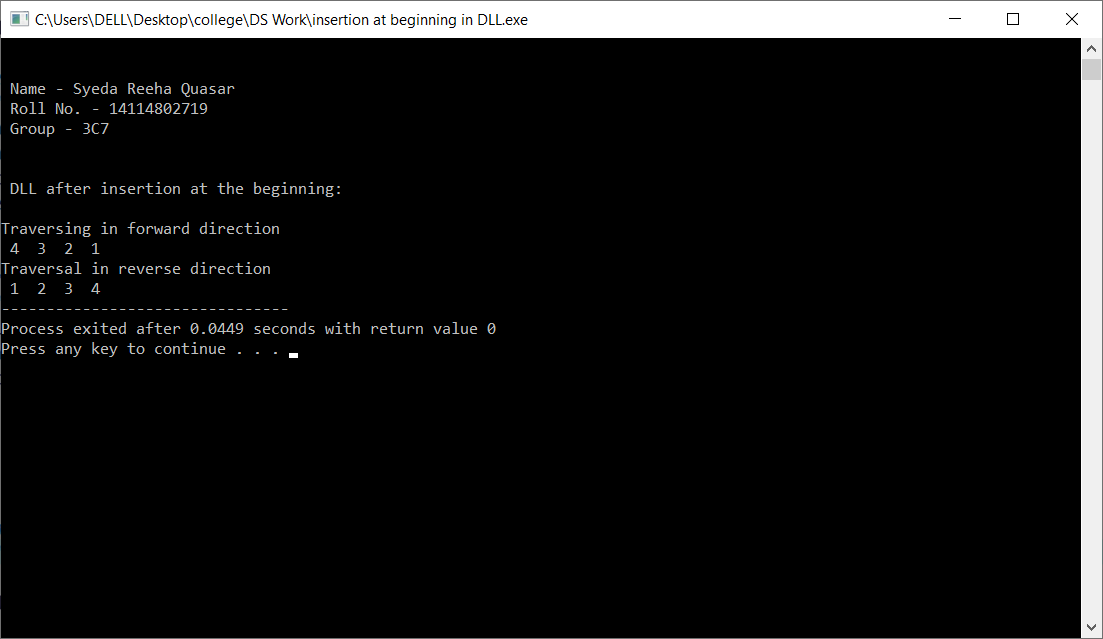
printf("\n DLL after insertion at the beginning: \n");

printList(head); // printing DLL

return 0;

}

**OUTPUT**

****

**Deletion from the end**

## **Source code:**

// deletion from the end doubly linked list

//required libraries

#include <stdio.h>

#include <stdlib.h>

// doubly linked list declaration

struct Node {

int data;

struct Node\* next; // Pointer to next node

struct Node\* prev; // Pointer to previous node

};

//function for insertion at the beginning

void insertAtBeginning(struct Node\*\* head\_ref, int newHeadData)

{

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node)); // new node

// assigning value to the new node and assigning values to its pointers

new\_node->data = newHeadData; //storing given data into new node

new\_node->next = (\*head\_ref); //dll next of node to head

new\_node->prev = NULL; // making new node head by pointing previous node to null

if ((\*head\_ref) != NULL) // checking if head is present or not

(\*head\_ref)->prev = new\_node; // changing poiter of headnode from null to new node

(\*head\_ref) = new\_node; // changing head to new node

}

void deletionAtEnd(struct Node\* node)

{

//traversing the list to find second lastnode

while (node -> next -> next != NULL) {

node = node->next;

}

node -> next = NULL; // changing second last node pointer to null

}

// printing the DLL

void printList(struct Node\* node)

{

struct Node\* last; // declaring a new node for reverse traversal

// traversal in forward direction

printf("\nTraversing in forward direction \n");

while (node != NULL) {

printf(" %d ", node->data);

last = node;

node = node->next;

}

//traversal in reverse direction

printf("\nTraversal in reverse direction \n");

while (last != NULL) {

printf(" %d ", last->data);

last = last->prev;

}

}

int main()

{

// my info

printf("\n\n Name - Syeda Reeha Quasar \n Roll No. - 14114802719 \n Group - 3C7 \n\n");

struct Node\* head = NULL; // declaring head as null

// all these elements are inserted in the beginning

insertAtBeginning(&head, 1); //inserting 1

insertAtBeginning(&head, 2); //inserting 2

insertAtBeginning(&head, 3); //inserting 3

insertAtBeginning(&head, 4); //inserting 4

printf("\n DLL before dletion: \n");

printList(head); // printing DLL

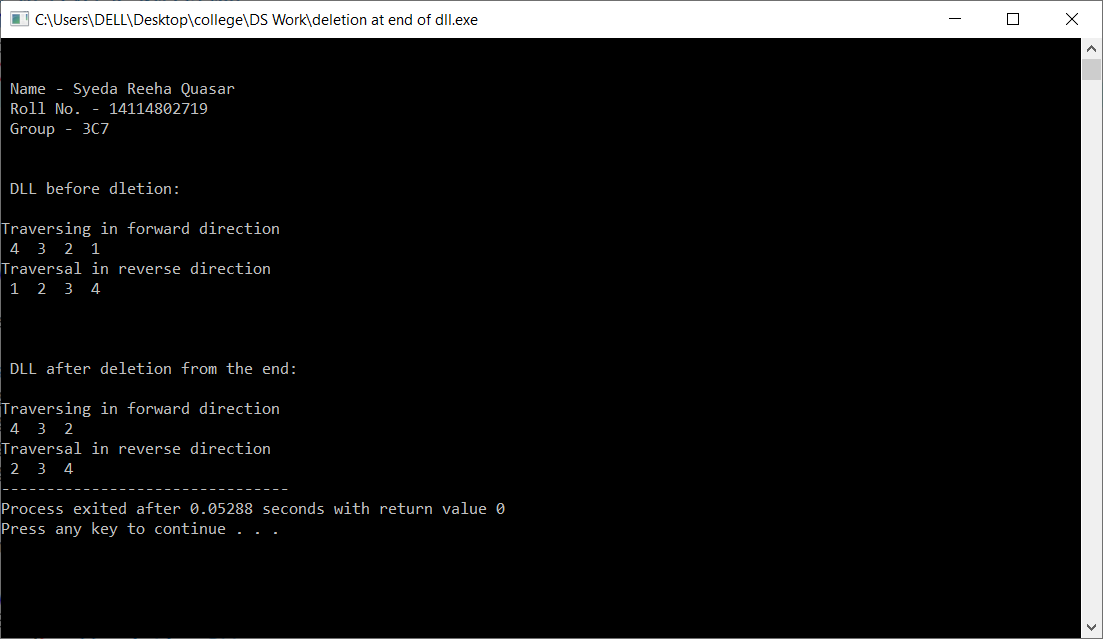
printf("\n\n\n\n DLL after deletion from the end: \n");

deletionAtEnd(head);

printList(head); // printing DLL

return 0;

}

**OUTPUT** 

**Viva Questions**

**Q1. What is a memory efficient doubly linked list?**

Ans.

A memory efficient version of Doubly Linked List can be created using only one space for address field with every node. This memory efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bitwise XOR operation to save space for one address.

**Q2. What is the advantage of doubly linked list?**

Ans.

Advantages of Doubly Linked List. A DLL can be traversed in both forward and backward direction.

The delete operation in DLL is more efficient if pointer to the node to be deleted is given. We can quickly insert a new node before a given node.

**Q3. Why is a doubly linked list more useful than a singly linked list?**

Ans.

1. A DLL can be traversed in both forward and backward direction.
2. The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node

**Q4. What is the difference between LinkedList and doubly linked list?**

Ans.

The main difference between singly linked list and doubly linked list is the ability to traverse. On the other hand doubly linked list maintains two pointers, towards next and previous node, which allows you to navigate in both direction in any linked list.

