EXPERIMENT - 8

Data Structures

Aim

Implement a Linear Queue using Linked List and Perform following operations:
Insert, Delete, and Display the queue elements

Syeda Reeha Quasar
14114902719
3C7

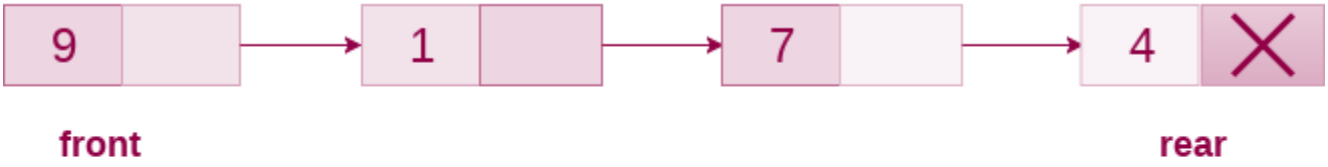
Name - Syeda Reeha Quasar **Roll no. –** 14114802719 **Group –** C7

EXPERIMENT – 8

AIM: Implement a Linear Queue using Linked List and Perform following operations: Insert, Delete, and Display the queue elements.

THEORY

The storage requirement of linked representation of a queue with n elements is o(n) while the time requirement for operations is O(1).
In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.
In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.
Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.
The linked representation of queue is shown in the following figure.



Linked Queue

Operation on Linked Queue

There are two basic operations which can be implemented on the linked queues. The operations are Insertion and Deletion.

Insertion

The insert operation append the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

ALGORITHM

- **Step 1:** Allocate the space for the new node PTR
- **Step 2:** SET PTR -> DATA = VAL

- **Step 3:** IF FRONT = NULL
  SET FRONT = REAR = PTR
  SET FRONT -> NEXT = REAR -> NEXT = NULL
  ELSE
  SET REAR -> NEXT = PTR
  SET REAR = PTR
  SET REAR -> NEXT = NULL
  [END OF IF]
- **Step 4:** END

Deletion

Deletion operation removes the element that is first inserted among all the queue elements. Firstly, we need to check either the list is empty or not. The condition front == NULL becomes true if the list is empty, in this case , we simply write underflow on the console and make exit.

Algorithm

- Step 1: IF FRONT = NULL
  Write " Underflow "
  Go to Step 5
  [END OF IF]
- Step 2: SET PTR = FRONT
- Step 3: SET FRONT = FRONT -> NEXT
- Step 4: FREE PTR
- Step 5: END

Source code:

```
// program for queue implementation using linked list

#include <iostream>

using namespace std;


// Node for LL declaration -> node for storing Queue elements

struct node {

        int data;

        node *next;

};

// front and rear pointers declaration

node *front = NULL;

node *rear = NULL;
```

```
// function for inserting the value into linked list
void enqueue(int value){
	node *ptr; //ptr for traversal in LL for queue
	ptr = new node;
	//checking whether queue is full
	if (ptr == NULL) {
		cout<<"Queue is full!"<<endl;
		return;
	}
	//inserting queue elements
	else {
		ptr -> data = value; // putting value of data as value given
		// if no front elements then insert ptr as front
		if (front == NULL) {
			front = ptr; // making front as ptr
			rear = ptr; // making rear as ptr
			front -> next = NULL; // pointing to null as only 1 element
			rear -> next = NULL;
		}
		// insertion at rear end... elements already in linked list
		else {
			rear -> next = ptr; // changing rear next to ptr
			rear = ptr; // inserting ptr containing the new node to the rear
			rear -> next = NULL; // making the next of rear as null
		}
	}
}

// function to dequeue or delete elemets from queue
```

```
void dequeue(){
	node *ptr; //ptr for traversal

	// condition when queue is empty
	if (front == NULL) {
		cout<<"queue is empty!"<<endl;
		return;
	}

	// removing front from queue
	else {
		ptr = front; // placing ptr aat front as front to be deleted
		front = front -> next; //changing front to front next to make second first element inserted as front
		delete ptr; // deleting the ptr
	}
}

// function to display the LL
void display(){
	struct node *ptr; //ptr for traversal
	ptr = front; // starting point of current ptr to front

	// condition if list is empty
	if (front == NULL) {
		cout<<"Queue is Empty!"<<endl; // checking if front exists
	}

	// traversing and printing list
	else {
```

```
            // traversing till we reach end of the linked list i.e. reaches rear
            while (ptr != NULL) {
                cout << ptr -> data << " "; // diplaying current node data
                cout << "-> "; // decorating the output
                ptr = ptr -> next; // moving to next
            }
        }
        cout << "NULL"; // adding null to the LL
}

int main(){
        int input; // variable for taking menu input
        cout << "Syeda Reeha Quasar \n 14114802719 \n C7"; // my info
        // menu for operation can be performed
        // do while used to execute the program atleast once
        do{
            cout<<endl;
            cout<<"1.Enqueue"<<endl; // 1st option for inserting elements
            cout<<"2.Dequeue"<<endl; // 2nd for deleting the elements from queue -> front is
deleted
            cout<<"3.Display"<<endl; // 3 for displaying the queue
            cout<<"0.Exit"<<endl;    // exiting the menu

            cin >> input; // taking input from user

            //checking what the input is and executing the desired operation
            if (input == 1) {
                cout<<"Enter Value to be inserted:"<<endl;
                int value;
                cin >> value; // taking value to be inserted from the user
```

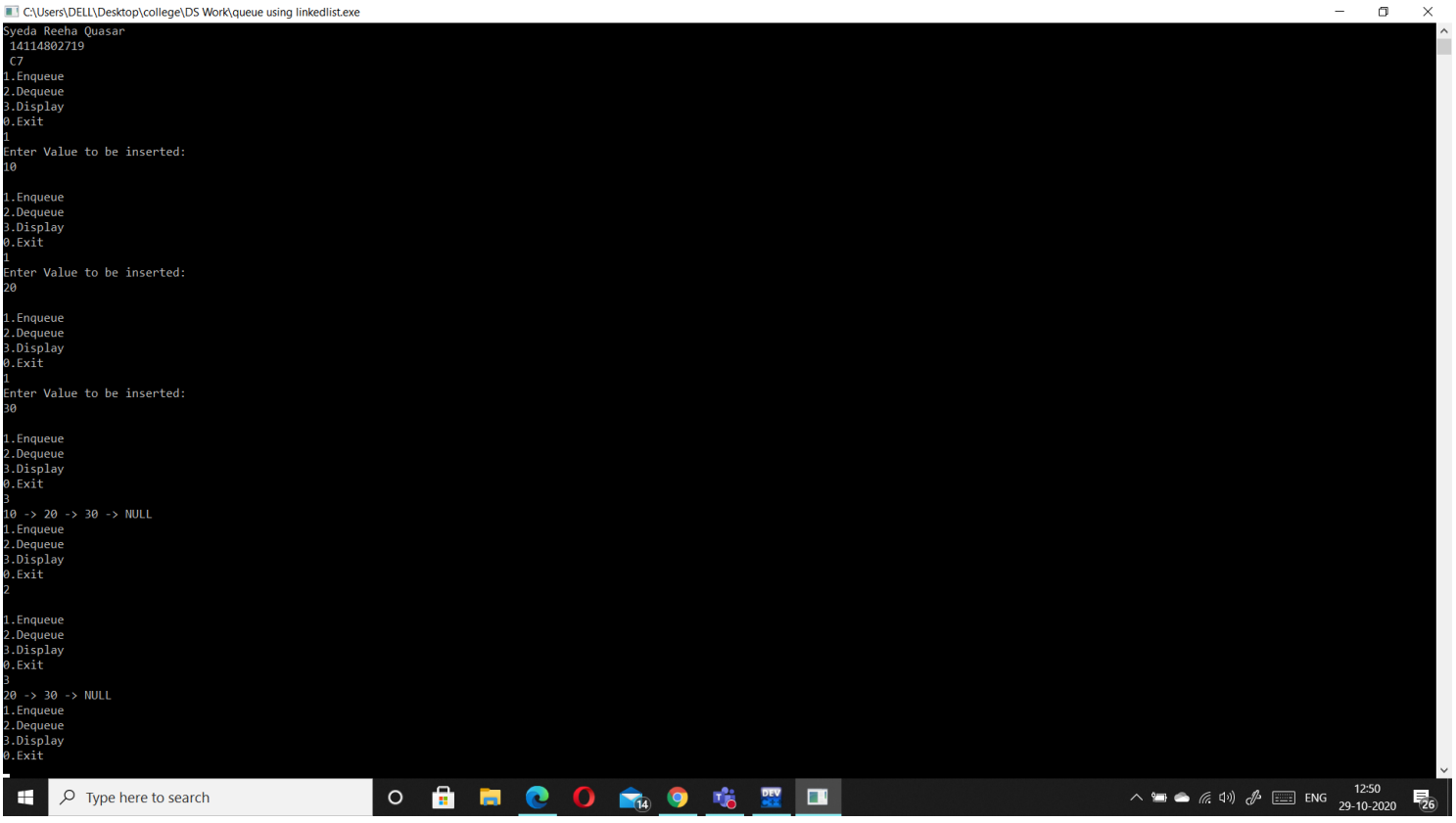
```
                enqueue(value);
        }

        if (input == 2) {
                dequeue();
        }

        if (input == 3) {
                display();
        }

    } while (input != 0);
    return 0;
}
```

OUTPUT



VIVA VOICE

Q1. In linked list implementation of queue, if only front pointer is maintained, which of the following operation take worst case linear time?

a) Insertion

b) Deletion

c) To empty a queue

d) Both Insertion and To empty a queue

Answer: d

Explanation: Since front pointer is used for deletion, so worst time for the other two cases.

Q.2 In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into a NONEMPTY queue?

a) Only front pointer

b) Only rear pointer

c) Both front and rear pointer

d) No pointer will be changed

Answer: b

Explanation: Since queue follows FIFO so new element inserted at last.

Q.3 In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into EMPTY queue?

a) Only front pointer

b) Only rear pointer

c) Both front and rear pointer

d) No pointer will be changed

Answer: c

Explanation: Since its the starting of queue, so both values are changed.