Course Name:	**Data Structures Laboratory**	**Semester:**	**III**
Date of Performance:	**01 / 09 / 2025**	**Batch No:**	**A1**
Faculty Name:	**Prof. Sushma Kadge**	**Roll No:**	**20**
Faculty Sign & Date:		**Grade/Marks:**	**/25**

Experiment No: 4
Title: Queue

Aim and Objective of the Experiment:

To understand the concept of queue.
Write a program for simple/linear queue using Linked list.
Given A[] = {11,33,55,10,66}
Perform enqueue, dequeue operations and display the queue contents.
Repeat the same operations and modify the code to implement Circular/ Priority/Double Ended queue type.

COs to be achieved:

CO1 : Understand and implement the different data structures used in problem solving
CO2: Apply linear and non-linear data structure in application development

Books/Journals/Websites referred:

1. GeeksforGeeks

Tools required:

DEV C/C++ compiler/ Code blocks C compiler

Theory:

A queue is a first-in, first-out (FIFO) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other end called the front. Like stacks, queues can be implemented by using either arrays or linked lists.

Implementation details:

1. Enlist all the Steps followed and various options explored
2. Explain your program logic and methods used.
3. Explain the Importance of the approach followed by you

=>

1. Implemented linear queue using linked list (`enqueue`, `dequeue`, `display`).
2. Modified program to implement circular queue using arrays.
3. Linked list avoids size limit; circular queue improves space use.

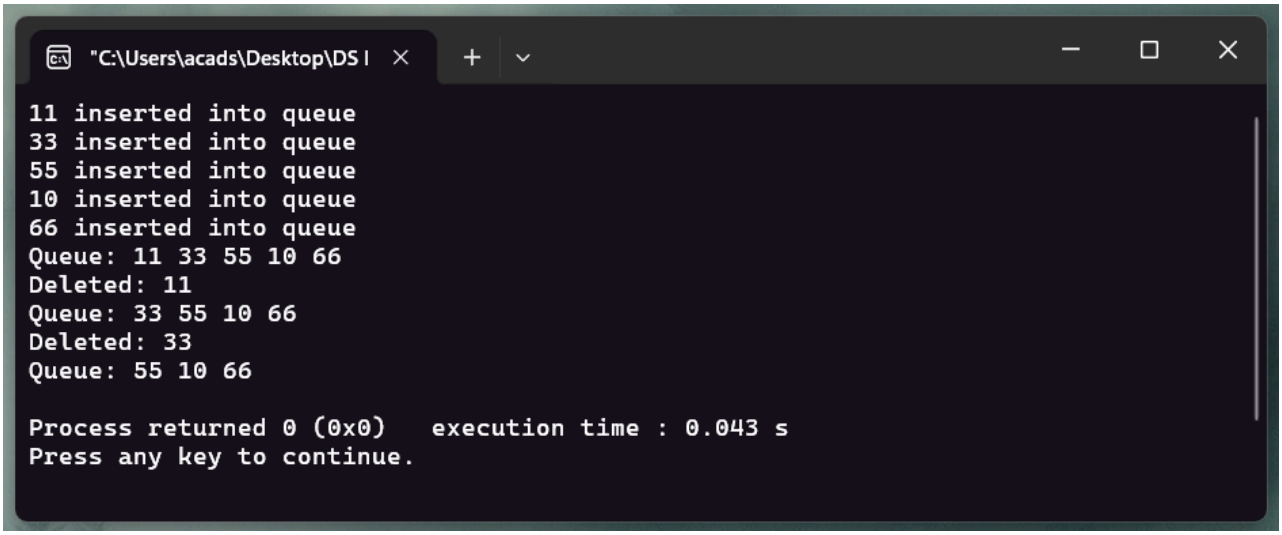
C/C++ Code implemented:

```c
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *next;
};

struct Node *front = NULL, *rear = NULL;

void enqueue(int val)
{
    struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = val;
    temp->next = NULL;
    if(front == NULL && rear == NULL)
    {
        front = rear = temp;
    }
    else
    {
        rear->next = temp;
        rear = temp;
    }
    printf("%d inserted into queue\n", val);
}

void dequeue()
{
    if(front == NULL)
    {
        printf("Queue is empty, cannot delete\n");
        return;
    }
    struct Node *temp = front;
    printf("Deleted: %d\n", temp->data);
    front = front->next;
    if(front == NULL)
        rear = NULL;
    free(temp);
}

void display()
{
    struct Node *temp = front;
    if(front == NULL)
    {
        printf("Queue is empty\n");
        return;
```

C/C++ Code implemented:

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

int main()
{
    int arr[] = {11, 33, 55, 10, 66};
    int n = 5, i;

    for(i = 0; i < n; i++)
        enqueue(arr[i]);

    display();

    dequeue();
    display();

    dequeue();
    display();

    return 0;
}
```

Output/ program results after execution:



Post Lab Subjective/Objective type Questions:

1. Write applications of the queue.

Ans:

1. **CPU scheduling** (process scheduling in operating systems).
2. **Printer management** (jobs handled in order).
3. **Disk scheduling** in operating systems.
4. **Network data packets** transmission.
5. **Call center systems** (calls answered in order).
6. **Breadth-First Search (BFS)** in graph/tree traversal.
7. **Order processing systems** in supermarkets and ticket counters.

Conclusion:

In this experiment, we successfully implemented a linear queue using a linked list and performed fundamental operations like enqueue, dequeue, and display. We also explored the modification to a circular queue to optimize memory utilization. Using a linked list for the linear queue allowed dynamic memory allocation, removing the size limitation of arrays, while the circular queue demonstrated efficient reuse of storage space.

Signature of faculty in-charge with Date: