Khwopa College of Engineering

DSA Lab Sheet

II Year/ II Part

Faculty: Computer and Electronics

Labsheet #4

Objectives:

1. Implementation of Linear Queue and Circular Queue using arrays.

Theory:

1. Linear Queue

A linear queue is a data structure that follows the **FIFO (First In First Out)** principle. It uses an array where insertion (enqueue) happens at the rear and deletion (dequeue) happens at the front.

Operations:

1. **Enqueue:** Insert element at rear**.**
2. **Dequeue**: Remove element from front.
3. **Peek**: Return the front element.
4. **IsFull / IsEmpty** checks.

Limitation:

Even after dequeuing, we can't reuse the freed space unless we shift all elements — inefficient!

2. Circular Queue

A circular queue connects the end of the array back to the front to form a circle. It solves the limitation of unused space in the linear queue.

Operations:

Same as linear, but the rear and front are updated using:

```
rear = (rear + 1) % SIZE;

front = (front + 1) % SIZE;
```

Advantages:

1. Efficient use of memory.
2. Circular nature allows wrap-around.

Objective #1: Linear Queue using Arrays

```
#Source Code

#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int front = -1, rear = -1;

void enqueue(int value) {

    if (rear == SIZE - 1)

        printf("Queue is Full\n");

    else {

        if (front == -1) front = 0;

        rear++;

        queue[rear] = value;

        printf("Inserted %d\n", value);

    }

}

void dequeue() {

    if (front == -1 || front > rear)

        printf("Queue is Empty\n");

    else {

        printf("Deleted: %d\n", queue[front]);

        front++;

    }

}

void display() {

    if (front == -1 || front > rear)

        printf("Queue is Empty\n");

    else {

        printf("Queue: ");

        for (int i = front; i <= rear; i++)

            printf("%d ", queue[i]);

        printf("\n");

    }

}

int main() {
    enqueue(10);

    enqueue(20);

    enqueue(30);

    display();

    dequeue();

    display();

    return 0;

}
```

Assignment #1: Run the given code and discuss the output with your teacher.

Assignment #2: Add peek(), isEmpty() and count() functions to the given queue. Where peek displays first element of the queue, isEmpty checks if the queue is empty and count counts the number of element in the queue.

Assignment #3: Write a function to display the queue in reverse order (without modifying the queue).

Assignment #4: Allow user to enter queue size dynamically.

Objective #2: Linear Queue using Arrays

```
#Source Code

#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int front = -1, rear = -1;

void enqueue(int value) {

    if ((rear + 1) % SIZE == front)

        printf("Queue is Full\n");

    else {

        if (front == -1)

            front = rear = 0;

        else

            rear = (rear + 1) % SIZE;

        queue[rear] = value;

        printf("Inserted %d\n", value);

    }

}

void dequeue() {

    if (front == -1)

        printf("Queue is Empty\n");

    else {

        printf("Deleted: %d\n", queue[front]);

        if (front == rear)

            front = rear = -1;

        else

            front = (front + 1) % SIZE;

    }

}

void display() {

    if (front == -1)

        printf("Queue is Empty\n");

    else {

        printf("Queue: ");

        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear)
                break;
            i = (i + 1) % SIZE;
        }
        printf("\n");
    }
}
int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    display();
    dequeue();
    enqueue(50);
    enqueue(60); // Should show full if already full
    display();
    return 0;
}
```

Assignment #1: Run the given code and discuss the output with your teacher.

Assignment #2: Add peek() and isEmpty() functions to the given queue.

Assignment #3: Allow user to enter queue size dynamically.

Assignment #4: Write a function to display the queue in reverse order (without modifying the queue).