Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 02

Course Title: Data Structure lab

Course Code: CSE 106 **Section**: 223(D5)

Lab Experiment Name: Implement Queue using Arrays

<u>Student Details</u>

	Name	ID
1.	**Promod Chandra Das**	**231002005**

Lab Date	**:**	
Submission Date	**:**	
Course Teacher's Name	**:**	**Md.Shihab Hossain**

<u>Lab Report Status</u>

Marks: ………………………………… **Signature:.....................**

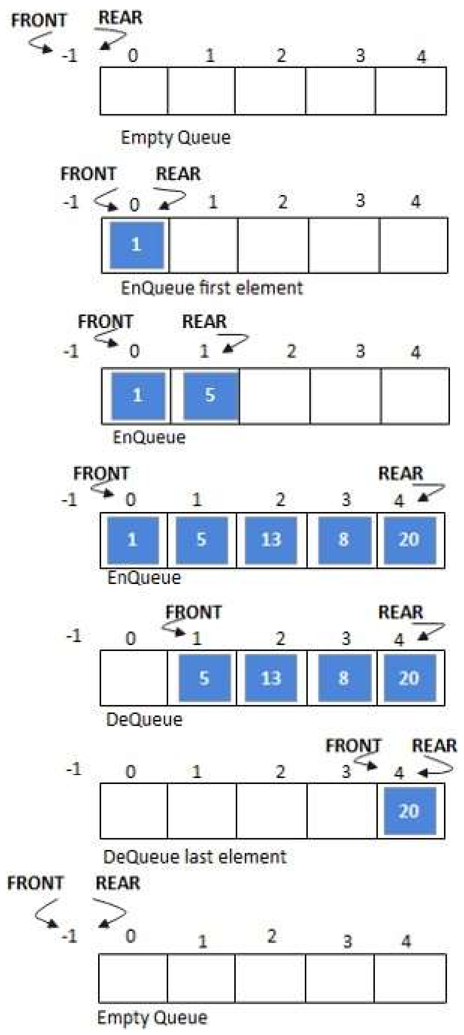
Comments:... **Date:..............................**

➢ Objective(s)

- To attain knowledge on the Queue data structure and how it works.
- To implement Queue using Arrays.

➢ Problem analysis

A queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR (also called tail), and the removal of existing element takesplace from the other end called as FRONT (also called head). This makes queue as FIFO (First In First Out) data structure, which means the element inserted first will be removed first.



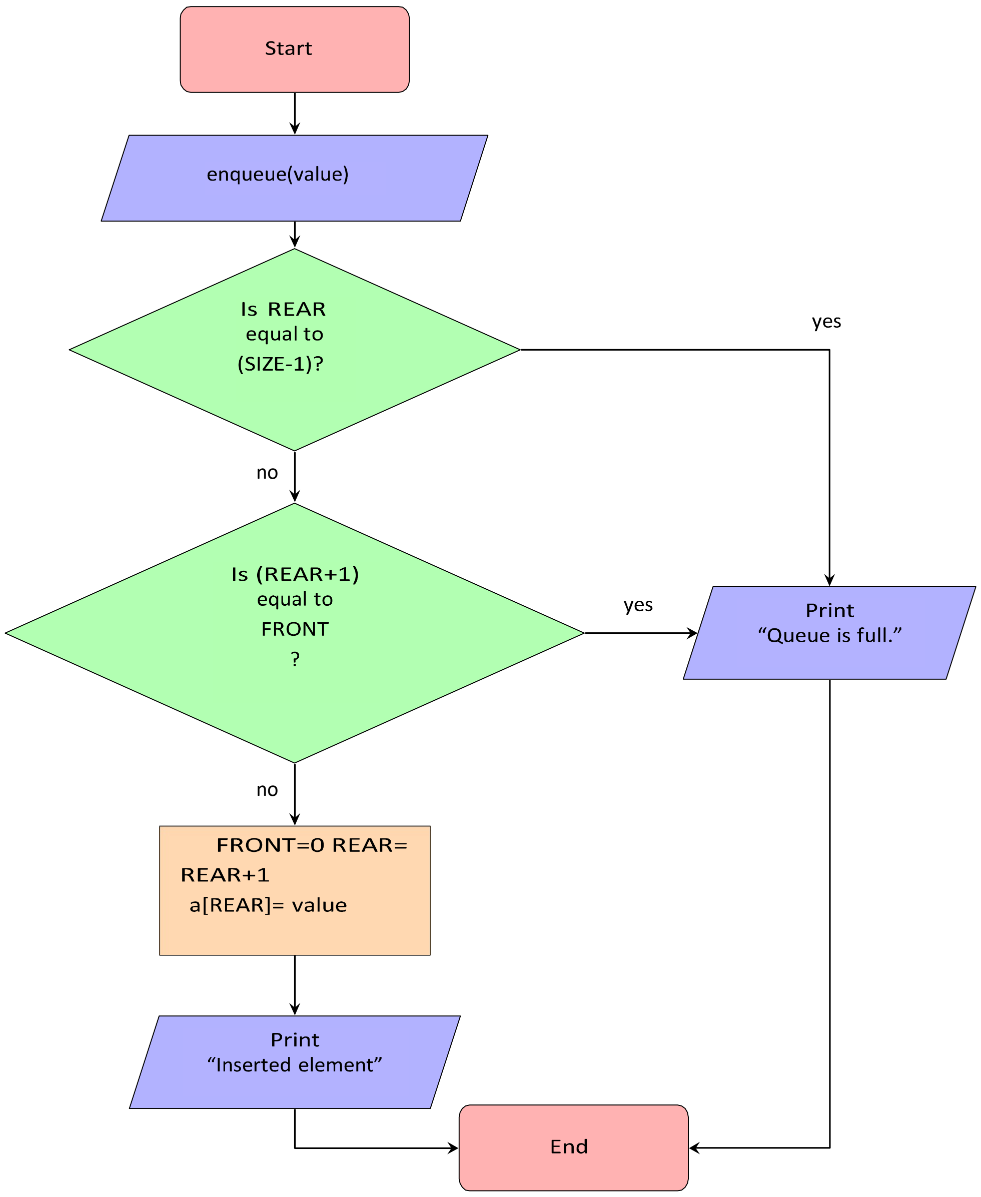
Algorithm

Algorithm 1: Enqueue Operation of Queue

```
Input: Element
/* Algorithm for enqueue operation                          */
```

```
if REAR == SIZE-1 then
  Print "Queue is full"
  return
  end
else
  if REAR + 1 == FRONT then
      Print "Queue is full"
      return
  end
  else
      FRONT = 0
      REAR = REAR + 1
      a[REAR] = value
      Print "Inserted element"
  end
end
```

➢ Flowchart



Algorithm 2: Dequeue Operation of Queue

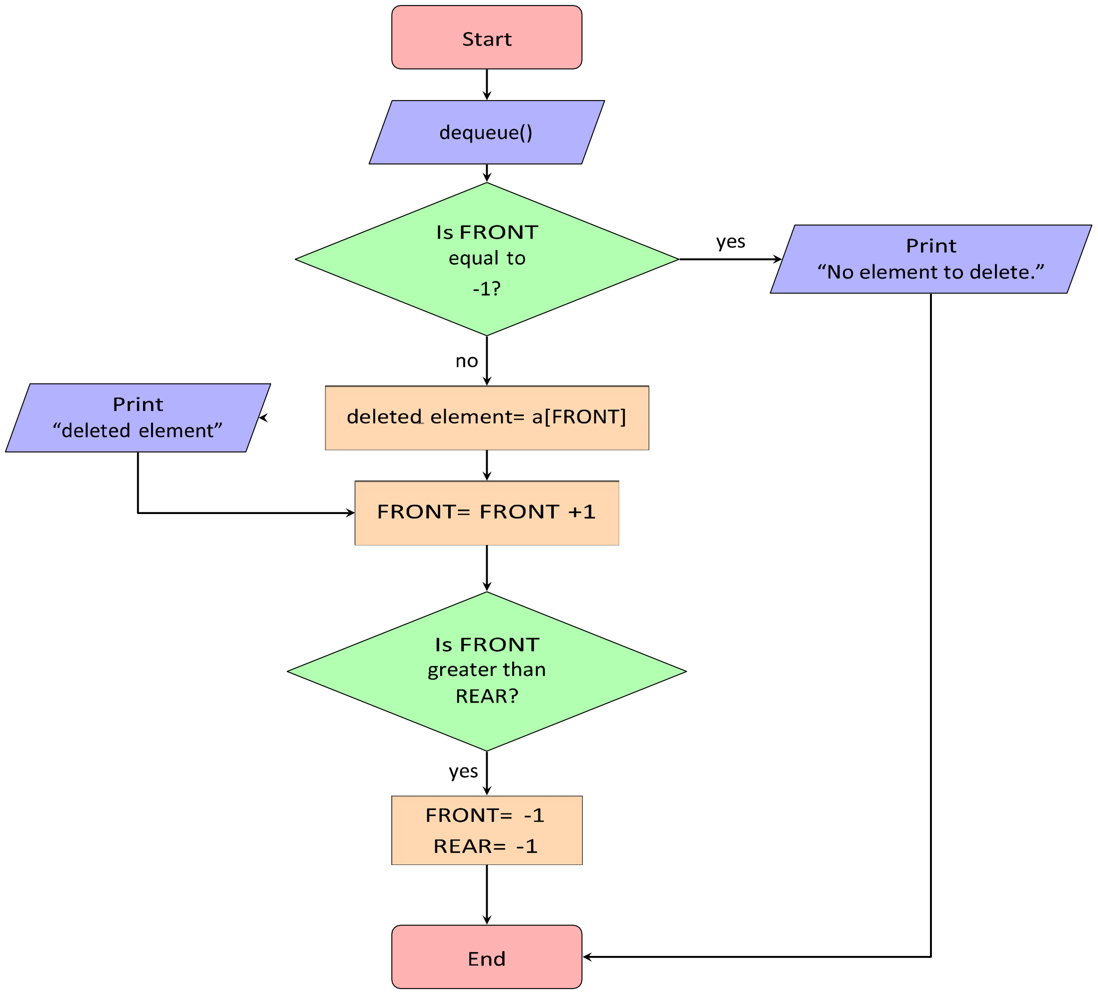
Output: Element

```
/* Algorithm for dequeue operation */
1 if FRONT == -1 then
```

```
2  Print "No element to delete."
3  return
4   end
5  else
6  Set Del_element = a[FRONT]
7  FRONT = FRONT+1
8  return Del_Element
9  if FRONT > REAR then
10   FRONT = -1
11   REAR = -1
12 end
```

```
13 end
```

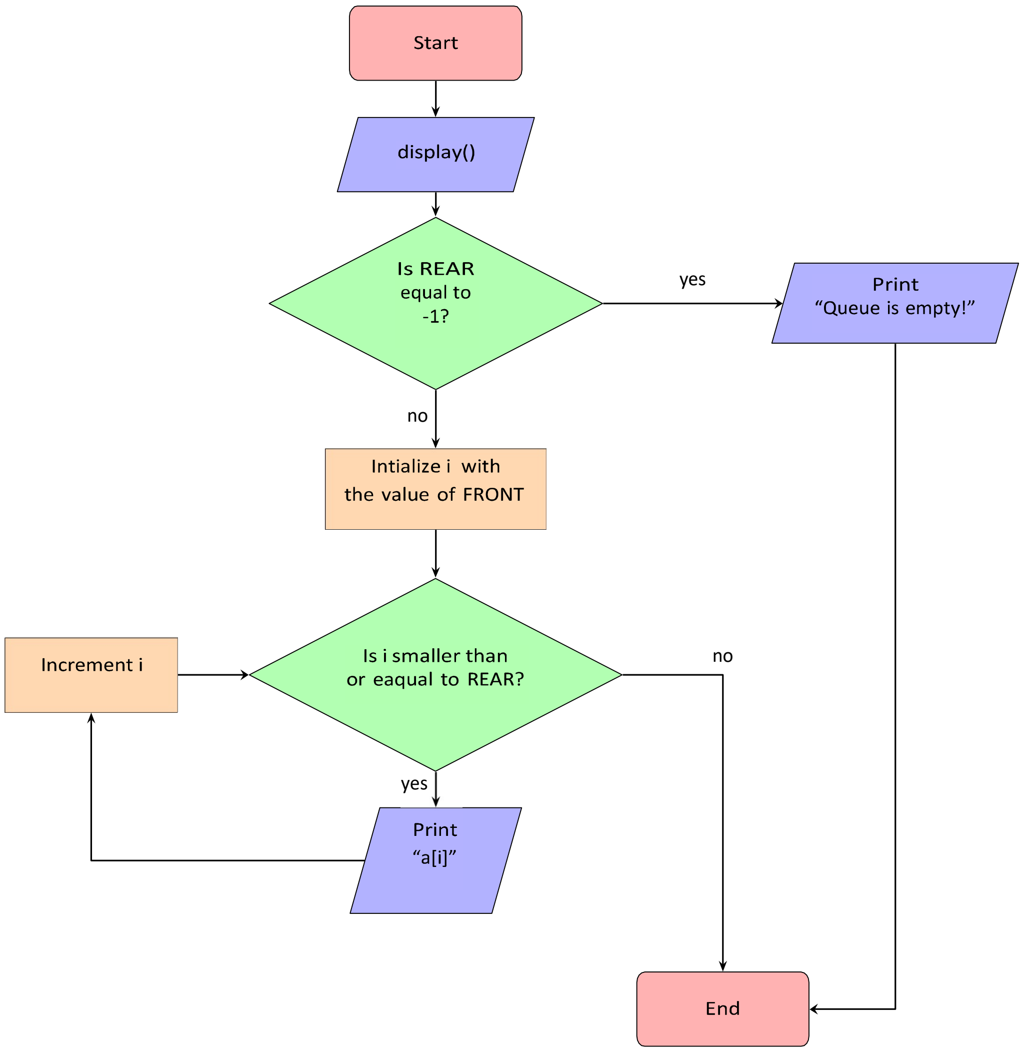
- Flowchart



Algorithm 3: Display Operation of Queue

```
   /* Algorithm for display operation                                   */
1 if REAR = -1 then
2     print "Queue is empty!"
3     return
4  end
5 else
6     Print "Elements in the queue"
7     for i = FRONT to REAR do
8         print "a[i]"
9     end
10 end
```

➢ **Flowchart**



2 Implementation in C

```
#include <stdio.h>
#include<stdlib.h>
#define size 5
int a[size],i,j,front=-1,rear=-1;

void enqueue(int value){
    if(rear==size-1)
        printf( "Queue is full \n");
```

```c
    else
    {
        if (rear+1==front)
            printf( "Queue is full.\n");
        else{
            front=0;
            rear++;
            a[rear]=value;
            printf("Inserted element: %d\n",a[rear]);
        }
    }

}


void dequeue(){
    if(front==−1){
        printf("No element to delete.\n");
    }

    else{
        printf("Deleted element: %d\n",a[front]);
        front=front+1;
        if(front>rear)
            front=rear=−1;
    }
}

void display(){
    if(rear==-1){
        printf("Queue is empty!\n");
    }
  else{
        printf("Elements in the queue: ");
        for(int i=front;i<=rear;i++){
            printf("%d ",a[i]);
        }
    }
}
int main(int argc, char const *argv[]) {
    int ch,val;
    do{
    printf("\nMenu");
    printf("\n1. ENQUEUE");
    printf("\n2. DEQUEUE ");
    printf("\n3. DISPLAY QUEUE");
    printf("\n4. Exit");
    printf("\nEnter your choice 1 to 4=");
    scanf("%d",&ch);
    switch (ch)
    {
        case 1:
                printf("Enter the value to be inserted=");
                scanf("%d",&val);
                enqueue(val);
```

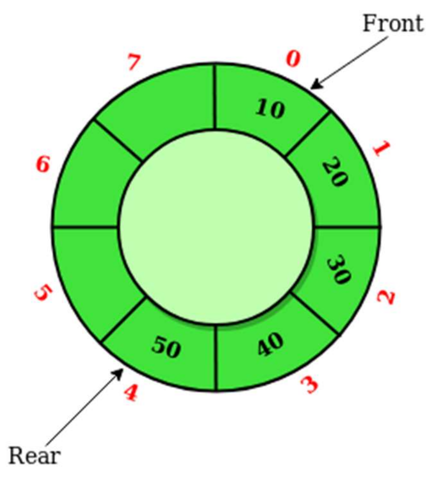
```
                    break;
            case 2:
                    dequeue();

                    break;

            case 3:

                    display();
                    break;
            case 4:
                    exit(0);
                    break;

            default:
                    printf("Invalid choice!");
                    break;
        }

    }while (ch<=3);

    return 0;
}
```

- Input/Output (Compilation, Debugging & Testing)

Input	Output
Choice: 3(initially)	Queue is empty.
Choice: 2(initially)	No element to delete.
Choice: 1 Enter the value to be inserted: 1	Inserted element: 1
Choice: 1 Enter the value to be inserted: 5	Inserted element: 5
Choice: 1 Enter the value to be inserted: 13	Inserted element: 13
Choice: 3	Elements in the queue: 1 5 13
Choice: 1 Enter the value to be inserted: 8	Inserted element: 8
Choice: 1 Enter the value to be inserted: 20	Inserted element: 20
Choice: 3	Elements in the queue: 1 5 13 8 20
Choice: 1 Enter the value to be inserted: 35	Queue is full
Choice: 3	Elements in the queue: 1 5 13 8 20
Choice: 2	Deleted element: 1
Choice: 3	Elements in the queue: 5 13 8 20
Choice: 2	Deleted element: 5
Choice: 2	Deleted element: 13
Choice: 3	Elements in the queue: 8 20
Choice: 2	Deleted element: 8
Choice: 3	Elements in the queue: 20
Choice: 2	Deleted element: 20
Choice: 3	Queue is empty
Choice: 2	No element to delete
Choice: 4	Press any key to continue.
Choice: 5	Invalid choice. Press any key to continue.

➢ Lab Exercise (Submit as a report)

- Implement the following circular queue using arrays



➢ <u>**Ans to the q no: 1**</u>

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

#define SIZE 8  // Define the maximum capacity of the queue

typedef struct {
   int items[SIZE];
   int front, rear;
} CircularQueue;

// Function to initialize the queue
void initQueue(CircularQueue *q) {
   q->front = -1;
   q->rear = -1;
}

// Function to check if the queue is full
int isFull(CircularQueue *q) {
    return ((q->rear + 1) % SIZE == q->front);
}

// Function to check if the queue is empty
int isEmpty(CircularQueue *q) {
    return (q->front == -1);
}

// Function to add an element to the queue
void enqueue(CircularQueue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % SIZE;
    q->items[q->rear] = value;
    printf("Enqueued %d\n", value);
}

// Function to remove an element from the queue
int dequeue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % SIZE;
    }
    printf("Dequeued %d\n", item);
    return item;
}

// Function to display the elements in the queue
```

```
void display(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    int i = q->front;
    do {
        printf("%d ", q->items[i]);
        i = (i + 1) % SIZE;
    } while (i != (q->rear + 1) % SIZE);
    printf("\n");
}

int main() {
    CircularQueue q;
    initQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    enqueue(&q, 40);
    enqueue(&q, 50);

    display(&q);

    dequeue(&q);
    display(&q);

    enqueue(&q, 60);
    display(&q);

    return 0;
}
```

➢ **Discussion & Conclusion:**

Implementing a circular queue using arrays efficiently utilizes storage by wrapping around the end to the beginning of the array. This approach ensures constant time complexity for enqueue and dequeue operations, provided the queue is not full or empty. It is particularly useful for buffering and managing circular data streams.