DAY 6 LAB

1. Write a program to implement Circular Queue with insertion and deletion operations.

```
#include <stdio.h>
#define SIZE 5
int items[SIZE];
int front = -1, rear = -1;
int isFull() {
   if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) {
      return 1;
   }
   return 0;
}
int isEmpty() {
   if (front == -1) {
      return 1;
   }
   return 0;
}
void enQueue(int element) {
   if (isFull()) {
      printf("\nQueue is full!");
   } else {
      if (front == -1) {
         front = 0;
      }
      rear = (rear + 1) % SIZE;
      items[rear] = element;
      printf("\nInserted element: %d", element);
   }
}
void deQueue() {
   int element;
   if (isEmpty()) {
      printf("\nQueue is empty!");
   } else {
      element = items[front];
      if (front == rear) {
         front = -1;
         rear = -1;
      } else {
         front = (front + 1) % SIZE;
      }
      printf("\nDeleted element: %d", element);
   }
}
void display() {
    int i;
    if (isEmpty()) {
        printf("\nQueue is empty!");
    } else {
        printf("\nFront: %d", front);
        printf("\nItems: ");
        for (i = front; i != rear; i = (i + 1) % SIZE) {
            printf("%d ", items[i]);
        }
        printf("%d", items[i]);
        printf("\nRear: %d", rear);
    }
}
int main() {
    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);
    display();
    deQueue();
    deQueue();
    display();
    enQueue(6);
    enQueue(7);
    display();
    return 0;
}
```

OUTPUT:

```
Inserted element: 1
Inserted element: 2
Inserted element: 3
Inserted element: 4
Inserted element: 5
Front: 0
Items: 1 2 3 4 5
Rear: 4
Deleted element: 1
Deleted element: 2
Front: 2
Items: 3 4 5
Rear: 4
Inserted element: 6
Inserted element: 7
Front: 2
Items: 3 4 5 6 7
Rear: 1
```

2. Write a program to implement Double Ended Queue with insertion and deletion operations.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
int deque[MAX_SIZE];
int front = -1, rear = -1;
void insertFront(int item) {
  if ((front == 0 && rear == MAX_SIZE - 1) || (front == rear + 1)) {
    printf("Queue Overflow\n");
    return;
  }
  if (front == -1) {
    front = 0;
    rear = 0;
  } else if (front == 0) {
    front = MAX_SIZE - 1;
  } else {
    front = front - 1;
  }
  deque[front] = item;
  printf("%d inserted at front\n", item);
}
void insertRear(int item) {
  if ((front == 0 && rear == MAX_SIZE - 1) || (front == rear + 1)) {
    printf("Queue Overflow\n");
    return;
  }
  if (rear == -1) {
    front = 0;
    rear = 0;
  } else if (rear == MAX_SIZE - 1) {
    rear = 0;
  } else {
    rear = rear + 1;
  }
  deque[rear] = item;
  printf("%d inserted at rear\n", item);
}
void deleteFront() {
  if (front == -1) {
    printf("Queue Underflow\n");
    return;
  }
  printf("%d deleted from front\n", deque[front]);
  if (front == rear) {
    front = -1;
    rear = -1;
  } else if (front == MAX_SIZE - 1) {
    front = 0;
  } else {
    front = front + 1;
  }
}
void deleteRear() {
  if (rear == -1) {
    printf("Queue Underflow\n");
    return;
  }
  printf("%d deleted from rear\n", deque[rear]);
  if (front == rear) {
    front = -1;
    rear = -1;
  } else if (rear == 0) {
    rear = MAX_SIZE - 1;
  } else {
    rear = rear - 1;
  }
}
void display() {
  int i;
  if (front == -1) {
    printf("Queue is empty\n");
    return;
  }
  printf("Elements in the Queue are: ");
  if (front <= rear) {
    for (i = front; i <= rear; i++) {
      printf("%d ", deque[i]);
    }
  } else {
    for (i = front; i < MAX_SIZE; i++) {
      printf("%d ", deque[i]);
    }
    for (i = 0; i <= rear; i++) {
      printf("%d ", deque[i]);
    }
  }
  printf("\n");
}
int main() {
  insertFront(1);
  insertRear(2);
  insertRear(3);
  display();
  deleteFront();
```

```
    display();
    insertFront(4);
    insertRear(5);
    display();
    deleteRear();
    display();
    return 0;
}
```

```
OUTPUT:
1 inserted at front
2 inserted at rear
3 inserted at rear
Elements in the Queue are: 1 2 3
1 deleted from front
Elements in the Queue are: 2 3
4 inserted at front
5 inserted at rear
Elements in the Queue are: 4 2 3 5
5 deleted from rear
Elements in the Queue are: 4 2 3
```

3. Write a program to implement Priority Queue with insertion and deletion operations.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
struct PriorityQueue {
    int items[MAX];
    int front;
    int rear;
};
void insert(struct PriorityQueue *q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue is full\n");
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear++;
        q->items[q->rear] = value;
    }
}
void delete(struct PriorityQueue *q) {
    if (q->front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Deleted item: %d\n", q->items[q->front]);
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
    }
}
void display(struct PriorityQueue *q) {
    if (q->rear == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements: ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
        printf("\n");
    }
}
int main() {
    struct PriorityQueue q;
    q.front = -1;
    q.rear = -1;
    insert(&q, 3);
    insert(&q, 5);
    insert(&q, 1);
    insert(&q, 2);
    display(&q);
    delete(&q);
    delete(&q);
    display(&q);
    return 0;
}
```

OUTPUT:

```
Queue elements: 3 5 1 2
Deleted item: 3
Deleted item: 5
Queue elements: 1 2
```