COMPUTER LAB

NAME: SOUMYADIP MAITY

ROLL NO.: 22053029

SEC: CSE-49

YEAR: 2023-24

ASSIGNMENT – 7

1. Write a menu driven program to implement queue operations such as Enqueue, Dequeue, Peek (display the front content), Display of elements, IsEmpty, IsFull using array.

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

#define MAX_SIZE 100

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

int isEmpty() {
    return (front == -1);
}

int isFull() {
    return (rear == MAX_SIZE - 1);
}

void enqueue(int data) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }

    if (isEmpty()) {
        front = 0;
    }

    rear++;
    queue[rear] = data;
    printf("%d enqueued successfully.\n", data);
}

void dequeue() {
    if (isEmpty()) {
        printf("Queue is empty. Cannot dequeue.\n");
        return;
    }

    int data = queue[front];
    front++;

    if (front > rear) {
        front = rear = -1;
    }

    printf("%d dequeued successfully.\n", data);
}

int peek() {
    if (isEmpty()) {
        printf("Queue is empty. Cannot peek.\n");
        return -1;
    }

    return queue[front];
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int choice, data;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. IsEmpty\n");
        printf("6. IsFull\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(data);
                break;
            case 2:
                dequeue();
                break;

            case 3:
                data = peek();
                if (data != -1) {
                    printf("Front element: %d\n", data);
                }
                break;

            case 4:
                display();
                break;

            case 5:
                if (isEmpty()) {
                    printf("Queue is empty.\n");
                } else {
                    printf("Queue is not empty.\n");
                }
                break;

            case 6:
                if (isFull()) {
                    printf("Queue is full.\n");
                } else {
                    printf("Queue is not full.\n");
                }
                break;

            case 7:
                printf("Exiting the program.\n");
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

2. Write a menu driven program to implement queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty using linked list.

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

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

struct Queue {
   struct Node* front;
   struct Node* rear;
};

struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = NULL;
   queue->rear = NULL;
   return queue;
}

int isEmpty(struct Queue* queue) {
   return (queue->front == NULL);
}

void enqueue(struct Queue* queue, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (isEmpty(queue)) {
      queue->front = newNode;
      queue->rear = newNode;
   } else {
      queue->rear->next = newNode;
      queue->rear = newNode;
   }

   printf("%d enqueued successfully.\n", data);
}

void dequeue(struct Queue* queue) {
   if (isEmpty(queue)) {
      printf("Queue is empty. Cannot dequeue.\n");
```

```
        return;
    }

    struct Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;
    free(temp);

    printf("%d dequeued successfully.\n", data);

    if (queue->front == NULL) {
        queue->rear = NULL;
    }
}

int peek(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot peek.\n");
        return -1;
    }

    return queue->front->data;
}

void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    struct Node* current = queue->front;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();

    int choice, data;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. IsEmpty\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(queue, data);
                break;

            case 2:
                dequeue(queue);
                break;

            case 3:
                data = peek(queue);
                if (data != -1) {
                    printf("Front element: %d\n", data);
                }
                break;

            case 4:
                display(queue);
                break;

            case 5:
                if (isEmpty(queue)) {
                    printf("Queue is empty.\n");
                } else {
                    printf("Queue is not empty.\n");
                }
                break;

            case 6:
                printf("Exiting the program.\n");
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

3. WAP using a function to reverse a queue by using stack.

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

#define MAX_SIZE 100

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

struct Queue {
   struct Node* front;
   struct Node* rear;
};
void initializeQueue(struct Queue* queue) {
   queue->front = NULL;
   queue->rear = NULL;
}

int isQueueEmpty(struct Queue* queue) {
   return (queue->front == NULL);
}

void enqueue(struct Queue* queue, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (isQueueEmpty(queue)) {
      queue->front = newNode;
      queue->rear = newNode;
   } else {
      queue->rear->next = newNode;
      queue->rear = newNode;
   }
}

int dequeue(struct Queue* queue) {
   if (isQueueEmpty(queue)) {
      printf("Queue is empty. Cannot dequeue.\n");
      return -1;
   }

   int data = queue->front->data;
   struct Node* temp = queue->front;
```

```
    queue->front = queue->front->next;
    free(temp);

    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    return data;
}

struct Stack {
    int arr[MAX_SIZE];
    int top;
};

void initializeStack(struct Stack* stack) {
    stack->top = -1;
}

int isStackEmpty(struct Stack* stack) {
    return (stack->top == -1);
}

void push(struct Stack* stack, int data) {
    if (stack->top == MAX_SIZE - 1) {
        printf("Stack is full. Cannot push.\n");
        return;
    }

    stack->arr[++(stack->top)] = data;
}

int pop(struct Stack* stack) {
    if (isStackEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
        return -1;
    }

    return stack->arr[(stack->top)--];
}

void reverseQueue(struct Queue* queue) {
    struct Stack stack;
    initializeStack(&stack);

    while (!isQueueEmpty(queue)) {
        int data = dequeue(queue);
        push(&stack, data);
    }

    while (!isStackEmpty(&stack)) {
        int data = pop(&stack);
        enqueue(queue, data);
    }
}

void display(struct Queue* queue) {
    if (isQueueEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    struct Node* current = queue->front;
    printf("Queue elements: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Queue queue;
    initializeQueue(&queue);

    enqueue(&queue, 7);
    enqueue(&queue, 1);
    enqueue(&queue, 3);
    enqueue(&queue, 11);
    enqueue(&queue, 2);

    printf("Original Queue:\n");
    display(&queue);

    reverseQueue(&queue);

    printf("Reversed Queue:\n");
    display(&queue);

    return 0;
}
```