# COMPUTER LAB

**NAME:** SOUMYADIP MAITY

**ROLL NO.:** 22053029

**SEC:** CSE-49

**YEAR:** 2023-24

# *ASSIGNMENT – 9*

## 1. *Write a menu-driven program to implement circular queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty, IsFull using Array.*

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
int front = -1, rear = -1;
int queue[MAX_SIZE];
int is_empty() {
return front == -1;
}
int is_full() {
return (rear + 1) % MAX_SIZE == front;
}
void enqueue(int data) {
if (is_full()) {
printf("Queue is full. Cannot enqueue.\n");
} else {
if (is_empty()) {
front = rear = 0;
} else {
rear = (rear + 1) % MAX_SIZE;
}
queue[rear] = data;
printf("Enqueued: %d\n", data);
}
}
void dequeue() {
if (is_empty()) {
printf("Queue is empty. Cannot dequeue.\n");
} else {
int data = queue[front];
if (front == rear) {
front = rear = -1;
} else {
front = (front + 1) % MAX_SIZE;
}
printf("Dequeued: %d\n", data);
}
}
void peek() {
if (is_empty()) {
printf("Queue is empty. Nothing to peek.\n");
} else {
printf("Peeked element: %d\n", queue[front]);
```

```
}
}void display() {
if (is_empty()) {
printf("Queue is empty. Nothing to display.\n");
} else {
printf("Queue elements: ");
int i = front;
do {
printf("%d ", queue[i]);
i = (i + 1) % MAX_SIZE;
} while (i != (rear + 1) % MAX_SIZE);
printf("\n");
}
}
int main() {
int choice, data;
while (1) {
printf("\nMenu:\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 the element to enqueue: ");
scanf("%d", &data);
enqueue(data);
break;
case 2:
dequeue();
break;
case 3:
peek();
break;
case 4:
display();
break;
case 5:
if (is_empty()) {
printf("Queue is empty.\n");
} else {
```

```
printf("Queue is not empty.\n");
}
break;
case 6:
if (is_full()) {
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 select a valid option.\n");
}
}
return 0;
}
```

## 2. WAP to implement a queue using stack data structure.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Stack {
int items[MAX_SIZE];
int top;
};
typedef struct Stack Stack;
void initialize(Stack* stack) {
stack->top = -1;
}
int is_empty(Stack* stack) {
return stack->top == -1;
}
int is_full(Stack* stack) {
return stack->top == MAX_SIZE - 1;
}
void push(Stack* stack, int value) {
if (!is_full(stack)) {
stack->items[++stack->top] = value;
} else {
printf("Stack overflow\n");
}
}
int pop(Stack* stack) {
```

```
if (!is_empty(stack)) {
return stack->items[stack->top--];} else {
printf("Stack is empty\n");
return -1;
}
}
void enqueue(Stack* stack1, int value) {
push(stack1, value);
}
int dequeue(Stack* stack1, Stack* stack2) {
if (is_empty(stack2)) {
while (!is_empty(stack1)) {
push(stack2, pop(stack1));
}
}
if (!is_empty(stack2)) {
return pop(stack2);
} else {
printf("Queue is empty\n");
return -1;
}
}
int main() {
Stack stack1, stack2;
initialize(&stack1);
initialize(&stack2);
int choice, data;
while (1) {
printf("\nMenu:\n");
printf("1. Enqueue\n");
printf("2. Dequeue\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter the element to enqueue: ");
scanf("%d", &data);
enqueue(&stack1, data);
break;
case 2:
data = dequeue(&stack1, &stack2);
if (data != -1) {
printf("Dequeued: %d\n", data);
}
break;
```

```
case 3:
printf("Exiting the program.\n");
exit(0);
default:printf("Invalid choice. Please select a valid option.\n");
}
}
return 0;
}
```

## 3. *WAP to implement a stack using queue data structure.*

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Queue {
int items[MAX_SIZE];
int front, rear;
};
typedef struct Queue Queue;
void initialize(Queue* queue) {
queue->front = queue->rear = -1;
}
int is_empty(Queue* queue) {
return queue->front == -1;
}
int is_full(Queue* queue) {
return (queue->rear + 1) % MAX_SIZE == queue->front;
}
void enqueue(Queue* queue, int value) {
if (!is_full(queue)) {
if (is_empty(queue)) {
queue->front = queue->rear = 0;
} else {
queue->rear = (queue->rear + 1) % MAX_SIZE;
}
queue->items[queue->rear] = value;
} else {
printf("Queue is full\n");
}
}
int dequeue(Queue* queue) {
if (!is_empty(queue)) {
int data = queue->items[queue->front];
if (queue->front == queue->rear) {
queue->front = queue->rear = -1;
} else {
```

```
queue->front = (queue->front + 1) % MAX_SIZE;}
return data;
} else {
printf("Queue is empty\n");
return -1;
}
}
void push(Queue* q1, Queue* q2, int value) {
if (is_empty(q1)) {
enqueue(q1, value)
;
} else {
while (!is_empty(q1)) {
enqueue(q2, dequeue(q1));
}
enqueue(q1, value);
while (!is_empty(q2)) {
enqueue(q1, dequeue(q2));
}
}
}
int pop(Queue* q1) {
if (!is_empty(q1)) {
return dequeue(q1);
} else {
printf("Stack is empty\n");
return -1;
}
}
int main() {
Queue q1, q2;
initialize(&q1);
initialize(&q2);
int choice, data;
while (1) {
printf("\nMenu:\n");
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter the element to push onto the stack: ");
scanf("%d", &data);
push(&q1, &q2, data);
```

```
break;
case 2:
data = pop(&q1);
if (data != -1) {
printf("Popped: %d\n", data);}
break;
case 3:
printf("Exiting the program.\n");
exit(0);
default:
printf("Invalid choice. Please select a valid option.\n");
}
}
return 0;
}
```