LAB3

Experiment 1

OBJECTIVE: To perform various operations in a queue using array implementation.

THEORY:

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. The element inserted first is removed first. The main operations on a queue are:

- 1. Enqueue (Insertion): Adds an element to the rear of the queue.
- 2. Dequeue (Deletion): Removes an element from the front of the queue.
- 3. Display: Shows all elements in the queue.
- 4. isEmpty: Checks if the queue is empty.
- 5. isFull: Checks if the queue is full.

Queue Representation in Array

- front points to the first element.
- rear points to the last inserted element.
- If rear == MAX 1, the queue is full.
- If front > rear or front == -1, the queue is empty.

ALGORITHM:

Enqueue Operation:

- 1. Check if the queue is full.
- 2. If not full, increment rear and insert the element.
- 3. If inserting the first element, set front = 0.

Dequeue Operation:

- 1. Check if the queue is empty.
- 2. If not empty, print and remove the front element.
- 3. Increment front.

Display Operation:

- 1. If empty, print "Queue is empty".
- 2. Otherwise, print elements from front to rear.

```
#include<stdio.h>
#include<conio.h>
                                                       void enqueue(int item) {
#define MAX 10
                                                           if (isFull()) {
                                                               printf("Queue is full\n");
int \ queue[MAX], \ front = -1, \ rear = -1;
                                                           else {
int isFull() {
                                                               if (front == -1) front = 0;
   return rear == MAX - 1;
                                                               rear++;
                                                               queue[rear] = item;
int isEmpty() {
   return\ front == -1 \mid \mid front > rear;
                                                       void dequeue() {
```

```
if (isEmpty()) {
                                                               printf("\n1. Enqueue\n2. Dequeue\n3.
       printf("Queue is empty\n");
                                                       Display\n4. Exit\nEnter your choice: ");
                                                               scanf("%d", &choice);
   else {
                                                               printf("\n");
       printf("Dequeued element: %d\n",
                                                               switch (choice) {
queue[front]);
                                                               case 1:
       front++;
                                                                  printf("Enter the element to enqueue:
                                                       ");
                                                                  scanf("%d", &item);
void display() {
                                                                  enqueue(item);
    if (isEmpty()) {
                                                                  break;
       printf("Queue is empty\n");
                                                               case 2:
                                                                  dequeue();
   else {
                                                                  break;
                                                              case 3:
       printf("Queue elements: ");
       for (int i = front; i \le rear; i++) {
                                                                  display();
           printf("%d ", queue[i]);
                                                                  break:
                                                               case 4:
       printf("\n");
                                                                  return 0;
                                                               default:
                                                                  printf("Invalid choice\n");
int main() {
                                                               printf("\nPress any key to continue...\n");
   int choice, item;
                                                               getch();
                                                           return 0;
   while (1) {
       system("cls");
                                                           getch();
       printf("Queue Operations using
Array \langle n'' \rangle;
```

Output:

```
C:\Users\Mahesh\Desktop\DSA in C\output\QueueAr
                                           C:\Users\Mahesh\Desktop\DSA in C\output\QueueAr
                                                                                      C:\Users\Mahesh\Desktop\DSA in C\output\Queu
Queue Operations using Array
                                            ueue Operations using Array
                                                                                      Queue Operations using Array

    Enqueue

                                              Enqueue

    Enqueue

2. Dequeue
                                            . Dequeue
                                                                                      Dequeue
Display
                                              Display
                                                                                      Display
. Exit
                                              Exit
                                            Enter your choice: 3
                                                                                      Enter your choice: 2
Enter your choice: 1
                                           Queue elements: 20 12 52 10 63 23
                                                                                     Dequeued element: 20
Enter the element to enqueue: 23
                                            ress any key to continue...
                                                                                      Press any key to continue...
 ress any key to continue...
```

CONCLUSION:

In this lab, we implemented queue operations using an array. The enqueue and dequeue functions worked as expected, following FIFO order. This demonstrates how queues can be efficiently managed using arrays.

Experiment 2

OBJECTIVE:

To perform various operations on a Queue with Linked List implementation.

THEORY:

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. The element inserted first is removed first. The main operations on a queue are:

- 1. Enqueue (Insertion): Adds an element to the rear of the queue.
- 2. Dequeue (Deletion): Removes an element from the front of the queue.
- 3. Display: Shows all elements in the queue.
- 4. isEmpty: Checks if the queue is empty.

Queue Representation using Linked List

- The queue is implemented using a linked list where each node contains data and a pointer to the next node.
- front points to the first node in the queue.
- rear points to the last node in the queue.
- If front == NULL, the queue is empty.

Algorithm

Enqueue Operation:

- 1. Create a new node.
- 2. If the queue is empty, set front = rear = new node.
- 3. Else, set rear->next = new node and update rear.

Dequeue Operation:

- 1. Check if the queue is empty.
- 2. If not empty, print and remove the front node.
- 3. Update front to the next node.

Display Operation:

- 1. If empty, print "Queue is empty".
- 2. Otherwise, traverse from front to rear and print elements.

```
#include<stdio.h>
                                                  int isEmpty() {
#include < conio. h >
                                                     return\ front == NULL;
#include<stdlib.h>
struct Node {
                                                  void enqueue(int item) {
                                                     Node* newNode =
   int data:
                                                  (Node*)malloc(sizeof(Node));
   struct Node* next:
};
                                                     newNode->data = item;
                                                     newNode -> next = NULL;
typedef struct Node Node;
                                                     if(rear == NULL) {
Node* front = NULL;
                                                         front = rear = newNode;
Node* rear = NULL;
                                                         rear->next = newNode;
```

```
rear = newNode:
                                                           int choice, item;
                                                           while (1) {
                                                              system("cls");
                                                              printf("\nQueue Operations using
void dequeue() {
                                                       Linked List\langle n \rangle n'');
   if (isEmpty()) {
                                                              printf("1. Enqueue\n2. Dequeue\n3.
                                                       Display\n4. Exit\nEnter your choice: ");
       printf("Queue is empty \n");
                                                              scanf("%d", &choice);
                                                              printf("\n");
   Node*temp = front;
                                                              switch (choice) {
   printf("Dequeued element: %d\n", front-
                                                              case 1:
>data);
                                                                  printf("Enter the element to enqueue:
                                                       ");
   front = front - next;
   if (front == NULL) rear = NULL;
                                                                  scanf("%d", &item);
                                                                  enqueue(item);
   free(temp);
                                                                  break:
                                                              case 2:
void display() {
                                                                  dequeue();
   if (isEmpty()) {
                                                                  break;
       printf("Queue is empty \n");
                                                              case 3:
                                                                  display();
       return;
                                                                  break;
   Node*temp = front;
                                                              case 4:
   printf("Queue elements: ");
                                                                  return 0;
   while (temp != NULL) {
                                                              default:
                                                                  printf("Invalid choice\n");
       printf("%d ", temp->data);
       temp = temp -> next;
                                                              printf("\nPress\ any\ key\ to\ continue...\n");
   printf("\n");
                                                              getch();
                                                           return 0;
int main() {
Output:
                                                                          Queue Operations using Linked List
                                      Queue Operations using Linked List
Queue Operations using Linked List
                                                                          1. Enqueue

    Enqueue

    Enqueue

                                                                          Dequeue
                                      Dequeue
Dequeue
                                                                          Display
                                      Display
Display
                                                                          4. Exit
                                        Exit
 . Exit
                                      Enter your choice: 2
                                                                          Enter your choice: 3
Enter your choice: 1
                                     Dequeued element: 23
                                                                          Queue elements: 52 14 25
Enter the element to enqueue: 25
                                      ress any key to continue...
                                                                           ress any key to continue...
```

CONCLUSION:

ress any key to continue...

In this lab, we implemented queue operations using a linked list. The enqueue and dequeue functions worked as expected, following FIFO order. This demonstrates how queues can be efficiently managed using linked lists.

Experiment 3

OBJECTIVE:

To perform various operations in a circular queue using array implementation.

THEORY:

A circular queue is a linear data structure that follows the First In, First Out (FIFO) principle but connects the end of the queue back to the front to utilize unused space efficiently.

Features of Circular Queue:

- 1. Efficient Space Utilization: Unlike a linear queue, it does not waste space after elements are dequeued.
- 2. Circular Structure: The rear wraps around when it reaches the last index.
- 3. Key Operations:
 - Enqueue: Adds an element at the rear.
 - Dequeue: Removes an element from the front.
 - Display: Shows all elements in the queue.
 - isFull: Checks if the queue is full.
 - isEmpty: Checks if the queue is empty.

Algorithm

Enqueue Operation:

- 1. Check if the queue is full.
- 2. If empty, set front = 0.
- 3. Increment rear circularly using (rear + 1) % MAX.
- 4. Insert the new element at rear.

Dequeue Operation:

- 1. Check if the queue is empty.
- 2. Print and remove the front element.
- 3. If only one element was left, reset front and rear to -1.
- 4. Otherwise, update front using (front + 1) % MAX.

Display Operation:

- 1. If empty, print "Queue is empty".
- 2. Traverse from front to rear circularly and print elements.

```
\begin{tabular}{ll} \#include &< stdio.h > & int is Empty() \{ \\ \#include &< stdib.h > & return (front == -1); \\ \#include &< conio.h > & \} \\ \#define MAX 10 & void enqueue (int value) \{ \\ int queue & [MAX], front = -1, rear = -1; \\ int is Full() & printf("Queue is full \n"); \\ int is Full() & return; \\ return (front == (rear + 1) % MAX); \\ \} & if (is Empty()) & \{ \\ front = 0; \\ \end{tabular}
```

```
int main() {
   rear = (rear + 1) \% MAX;
                                                           int choice, value;
   queue[rear] = value;
                                                           while (1) {
                                                               system("cls");
                                                               printf("Circular Queue
void dequeue() {
                                                       Operations:\n\n'');
                                                               printf("1. Enqueue\n2. Dequeue\n3.
    if (isEmpty()) {
                                                       Display \setminus n4. Exit \setminus n'');
       printf("Queue is empty \n");
                                                               printf("Enter your choice: ");
       return;
                                                               scanf("%d", &choice);
   printf("Dequeued element: %d\n",
                                                               switch (choice) {
queue[front]);
                                                                   case 1:
   if (front == rear) {
                                                                      printf("Enter value to enqueue:
       front = -1;
                                                       ");
       rear = -1;
                                                                       scanf("%d", &value);
                                                                       enqueue(value);
   } else {
       front = (front + 1) \% MAX;
                                                                       break:
                                                                   case 2:
                                                                       dequeue();
                                                                       break:
void display() {
                                                                   case 3:
   if (isEmpty()) {
                                                                       display();
       printf("Queue is empty\n");
                                                                       break;
                                                                   case 4:
                                                                       exit(0);
   printf("Queue elements: ");
                                                                   default:
   for (int i = front; i != rear; i = (i + 1) \%
                                                                      printf("Invalid choice\n");
MAX) {
                                                               printf("\nPress\ any\ key\ to\ continue...\n");
       printf("%d ", queue[i]);
                                                               getch();
   printf("%d\n", queue[rear]); // Print the last
element
                                                           return 0;
```

Output:

```
C:\Users\Mahesh\Desktop\DSA in C\ordinate
                                     C:\Users\Mahesh\Desktop\DSA in C\output\Que
C:\Users\Mahesh\Desktop\DSA in C\output\@
                                     Circular Queue Operations:
                                                                              Circular Queue Operations:
Circular Queue Operations:
                                     1. Enqueue
                                                                              1. Enqueue

    Enqueue

2. Dequeue
                                     Dequeue
                                                                                 Dequeue
3. Display
4. Exit
                                     3. Display
                                                                                 Display
                                     4. Exit
                                                                                 Exit
                                     Enter your choice: 3
Enter your choice: 1
                                                                              Enter your choice: 2
Enter value to enqueue: 64
                                     Queue elements: 23 45 96 47 64
                                                                              Dequeued element: 23
Press any key to continue...
                                     Press any key to continue...
                                                                              Press any key to continue.
```

CONCLUSION:

In this lab, we implemented a circular queue using an array. The enqueue and dequeue operations worked efficiently, demonstrating the advantage of circular queues in avoiding wasted space compared to linear queues.

Experiment 4

OBJECTIVE: To perform various operations in a priority queue, implementing both Min and Max Priority Queues using arrays.

THEORY:

A Priority Queue is a special type of queue in which elements are inserted based on their priority. The two types of priority queues are:

- 1. Min Priority Queue: The element with the lowest value has the highest priority and is dequeued first.
- 2. Max Priority Queue: The element with the highest value has the highest priority and is dequeued first.

Key Operations:

- Enqueue: Inserts an element in the correct position based on priority.
- Dequeue: Removes the highest (or lowest) priority element.
- Display: Shows elements in priority order.
- isEmpty: Checks if the queue is empty.

Algorithm

Enqueue (Insertion):

- Insert the element at the rear.
- Sort the queue based on priority (ascending for Min Queue, descending for Max Queue).

Dequeue (Deletion):

• Remove the first element (highest priority).

Display:

• Print elements from front to rear.

```
#include <stdio.h>
                                                      minRear++;
#include <conio.h>
#include <stdlib.h>
                                                      minQueue[minRear] = value;
#define MAX 10
                                                     for (int i = minRear; i > minFront &&
                                                  minQueue[i] < minQueue[i - 1]; i--) 
                                                         int temp = minQueue[i];
int minQueue[MAX], maxQueue[MAX];
int minFront = -1, minRear = -1;
                                                         minQueue[i] = minQueue[i - 1];
int maxFront = -1, maxRear = -1;
                                                         minQueue[i - 1] = temp;
int isEmpty(int front) {
   return\ front == -1;
                                                  void minDequeue() {
                                                      if (isEmpty(minFront)) {
                                                         printf("Min Priority Queue is empty\n");
void minEnqueue(int value) {
   if (minRear == MAX - 1)  {
                                                         return;
       printf("Min Priority Queue is full\n");
                                                     printf("Dequeued Min: %d\n",
       return;
                                                  minQueue[minFront]);
   if (isEmpty(minFront)) {
                                                      if (minFront == minRear) minFront =
       minFront = 0:
                                                  minRear = -1;
```

```
system("cls");
   else minFront++;
                                                          printf("\nPriority Queue
                                                   Operations:\n\n");
void maxEnqueue(int value) {
                                                          printf("1. Min Enqueue\n2. Min
   if(maxRear == MAX - 1) {
                                                   Dequeue\n3. Display Min Queue\n");
       printf("Max Priority Queue is full\n");
                                                          printf("4. Max Enqueue\n5. Max
                                                   Dequeue\n6. Display Max Queue\n7. Exit\n");
       return;
                                                          printf("Enter your choice: ");
   if (isEmpty(maxFront)) {
                                                          scanf("%d", &choice);
       maxFront = 0;
                                                          printf("\n");
                                                          switch (choice) {
   maxRear++;
                                                              case 1:
   maxQueue[maxRear] = value;
                                                                  printf("Enter value to enqueue
   for (int i = maxRear; i > maxFront &&
                                                   (Min Priority): ");
maxQueue[i] > maxQueue[i - 1]; i--) {
                                                                  scanf("%d", &value);
       int temp = maxQueue[i];
                                                                  minEnqueue(value);
       maxQueue[i] = maxQueue[i - 1];
                                                                  break:
       maxQueue[i - 1] = temp;
                                                              case 2:
                                                                  minDequeue();
                                                                  break:
                                                              case 3:
void maxDequeue() {
                                                                  display(minQueue, minFront,
   if (isEmpty(maxFront)) {
                                                   minRear);
       printf("Max Priority Queue is empty\n");
                                                                  break;
                                                              case 4:
       return;
                                                                  printf("Enter value to enqueue
   printf("Dequeued Max: %d\n",
                                                   (Max Priority): ");
maxQueue[maxFront]);
                                                                  scanf("%d", &value);
   if (maxFront == maxRear) maxFront =
                                                                  maxEnqueue(value);
maxRear = -1;
                                                                  break:
   else maxFront++;
                                                              case 5:
                                                                  maxDequeue();
                                                                  break;
void display(int queue[], int front, int rear) {
                                                              case 6:
   if (isEmpty(front)) {
                                                                  display(maxQueue, maxFront,
       printf("Queue is empty\n");
                                                   maxRear);
                                                                  break;
       return;
                                                              case 7:
   printf("Queue elements: ");
                                                                  exit(0);
   for (int i = front; i \le rear; i++) {
                                                              default:
       printf("%d ", queue[i]);
                                                                  printf("Invalid choice\n");
   printf("\n");
                                                          printf("\nPress any key to continue...\n");
                                                          getch();
                                                       return 0;
int main() {
   int choice, value;
   while (1) {
```

Output:

```
Priority Queue Operations:

    Min Enqueue

Priority Queue Operations:
                                       2. Min Dequeue
                                       Display Min Queue
1. Min Enqueue
                                       4. Max Enqueue
2. Min Dequeue
3. Display Min Queue
                                       5. Max Dequeue
4. Max Enqueue
5. Max Dequeue
                                       Display Max Queue
                                       Exit
5. Display Max Queue
                                       Enter your choice: 3
  Exit
Enter your choice: 1
                                       Queue elements: 12 45 52 63
Enter value to enqueue (Min Priority): 45
                                       Press any key to continue...
Press any key to continue...
Priority Queue Operations:
```

```
1. Min Enqueue
2. Min Dequeue
3. Display Min Queue
4. Max Enqueue
5. Max Dequeue
6. Display Max Queue
7. Exit
Enter your choice: 2
Dequeued Min: 12
Press any key to continue...
```

```
Priority Queue Operations:

1. Min Enqueue
2. Min Dequeue
3. Display Min Queue
4. Max Enqueue
5. Max Dequeue
6. Display Max Queue
7. Exit
Enter your choice: 4

Enter value to enqueue (Max Priority): 16
```

```
Priority Queue Operations:

1. Min Enqueue
2. Min Dequeue
3. Display Min Queue
4. Max Enqueue
5. Max Dequeue
6. Display Max Queue
7. Exit
Enter your choice: 6

Queue elements: 97 63 54 52 16

Press any key to continue...
```

```
Priority Queue Operations:

1. Min Enqueue
2. Min Dequeue
3. Display Min Queue
4. Max Enqueue
5. Max Dequeue
6. Display Max Queue
7. Exit
Enter your choice: 5

Dequeued Max: 97

Press any key to continue...
```

CONCLUSION:

In this lab, we implemented a Priority Queue using arrays. The Min Priority Queue removes the smallest element first, while the Max Priority Queue removes the largest element first. This approach is useful in scheduling and real-time processing applications.