Experiment 10

Circular queue

```
#include <stdio.h>
#include <stdlib.h>
# define max 6

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

void enqueue (int val);
void dequeue ();
void display ();

int main() {
int choice value;
while (1) {
  printf("--menu ---\n");
  printf("1. Enqueue \n");
  printf("2. Dequeue \n");
  printf("3. Display \n");
  printf("4. Exist \n");
  printf("Choose option:");
  scanf("%d", &choice);
switch (choice){
  case 1:
    printf("\n Enter value to be enqueued ");
    scanf("%d", &value);
    enqueue(value);
    break;
```

```
case 2:
deqeue ();
break;
case 3:
display ();
break;
case 4:
exit (0);
break;
default:
printf("Invalid Input !");
}
}
return 0;
}
void enqueue (int val){
if(front==0 && rear==max-1){
printf("Queue full! Cannot enqueue");
} else if (front == rear+1){
printf( Queue full ! Cannot enqueue");
}
else if(front==rear+1){
printf("Queue full
else{
if(front==-1 && rear ==-1){
front = rear = 0;
queue[rear]=val;
}else if [rear]==max-1 && front!=0;
rear = 0;
queue [rear]=val;
}else{
rear++;
```

```
queue [rear] = val;
}
printf("%d inserted successfully!",val);
}
}

void display() {
if (front == -1) {
printf("Queue empty! ");
}
else {
int i = front;
printf("queue Elements are: ");
while(1) {
printf("%d ", queue[i]);
if ( i == rear ){
break;
}
i = (i+1) % max;
}
}
}.
```

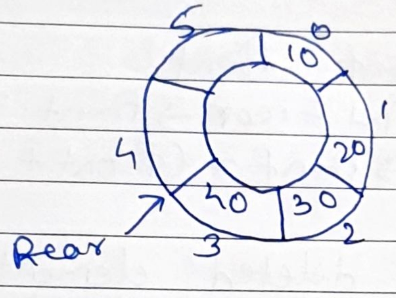
~~Experiment~~ ~~ass~~

Q1) Advantages of circular queue over normal queue.

→ 1). Efficient memory use:- A circular queue empty spaces. In a normal queue, once an el is removed, its spot remains empty, wasting memory.

2) Constant Time complexity:- Both insertion and deletion are performed in constant time. O(1) because pointers just move without need to shift elements.

3) Simple pointer operators:- Logic to move front & rear pointer is straight forward.



Circular queue.

Q2) Algorithms to insert in queue (enqueue)

⇒ adds element to rear ~~to~~ of queue.

Step1 - Start

Step2 - Check for overflow (queue full)

- if (front == 0 && ~~rear~~ == Max_size - 1)
- if (front = rear + 1)
- the queue is full if either of the conditions is true
- Display overflow error.

Step 3: Insert element at rear ++
- if front = rear == -1 → set both to 0
- insert at queue [rear]
- else → rear = (rear+1) to max
- insert at queue [rear]

Step 4: exit.

→ Algo to delete

Step 1: start
Step 2: Check for underflow
- if front == -1 → queue empty

Step 3: Retrieve the element.
- data = queue [front]

Step 4: increment front
- if front = rear → front = rear -1
- else → front = (front + 1) % max_size

Step 5 = Return deleted element.

Step 6: exit.

Q3) Applications of Queue
1) CPU & task scheduling
2) Data buffering
3) Breadth first search (BFS)
4) Pointer spooling.

Applications of circular queue.

1) Memory management
2) Traffic light systems
3) Resource pooling

Q4) Differ between stack & queue.

	Stack	Queue
Principle	Last in first out (LIFO)	First in first out (FIFO)
Pointers	Has only one active end pointer → TOP	Has two pointers pointing to first & last element → front and rear.
Operations	Push → add element POP → Remove element	Enqueue → add element. Dequeue → Remove element.
Data access	Only top is easily accessible	Only front & rear are involved.
Uses	function call stack, undo redo, expression evaluation.	CPU scheduling, Managing server requests, BFS

13/1