ASSIGNMENT- 03

NAME: M. Prasad

REG NO: 192311852

COURSE CODE: CSA 0389

COURSE NAME: DATA STRUCTURES

DATE: 21-Aug-2024

State the queue operation using following function calls size = 5. Enqueue(25), enqueue(37), enqueue(90), queue(), enqueue(15), enqueue(40), enqueue(12), dequeue(), Dequeue(), Dequeue(), dequeue().

To illustrate the queue operation for a queue of sizes 5 with given sequence of function calls, let's through each other :-

Initial Queue state :-

* The Queue is empty initially
* Maximum size of Queue : 5

Operations :-

1. Enqueue(25) :
   * Queue : '[25]'
   * Front = 0, Rear = 0
2. Enqueue(37) :
   * Queue [25, 37]
   * Front = 0, Rear = 1
3. Enqueue(90) :
   * Queue : [25, 37, 90]
   * Front = 0, Rear = 2
4. Dequeue() :
   * 25 is removed from queue
   * Queue : [37, 90]
   * front = 1, Rear = 2

5. Enqueue(15):

* Queue = [37, 90, 15]
* Front = 1, Rear = 3

6. Enqueue(40):

* Queue: [37, 90, 15, 40,]
* Front = 1, Rear = 4

7) Enqueue(12):

* Queue : [37, 90, 15, 40, 12]
* front = 1, Rear = 5

8. dequeue():

* 37 is removed from the queue
* Queue : [90, 15, 40, 12]
* front = 2, Rear 5

9. Dequeue()

* 90 is removed from queue
* Queue : [15, 40, 12]
* Front = 3, Rear = 15

10. Dequeue():

* 15 is removed from queue
* Queue : [12]
* front = 4, Rear = 5

dequeue():

HD is removed from queue

* queue: [12]

* front=5, rear=5

final queue state:

* The queue contains [12] after all operations are performed.

* front=5 rear=5

summary of operations:-

→ The operations performed show how elements are enqueued and dequeued from queue.

→ The dequed maximum size of never exceed and element are dequeued in the order they were enqueued following the first-in-first out [FIFO] principal.

2) write a c program to implement queue operations such as enqueue, dequeue and display.

```
#include <stdio.h>
#include <stdlib.h>
# define size 5
    struct queue{
        int items[size];
```

```
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

int isFull(struct Queue* queue) {
    if (queue->rear == size - 1)
        return 1;
    return 0;
}

int isEmpty(struct Queue* queue) {
    if (queue->front == -1 || queue->front > queue->rear)
        return 1;
    return 0;
}

void enqueue(struct Queue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue is full! Cannot enqueue %d\n", value);
    } else {
        if (queue->front == -1
```

```
    queue->front = 0;
    queue->rear++;
    queue->items[queue->rear] = value;
printf("Enqueued %d\n", value);
    }
}
    void dequeue(struct Queue* queue) {
        if (isempty(queue)) {
  printf("Queue is empty! cannot dequeue\n");
}else {
printf("dequeued %d\n", queue->items[queue->
    front]);
      queue->front++;
}}
    void display(struct Queue* queue) {
      if (isempty(queue)) {
      printf("Queue is empty!\n");} {
     else{
          printf("Queue:");
for(int i=queue->front; i<queue->rear; i++){
    printf("%d", queue->items[i]);
    }
    printf("\n");}
    }
```

```
int main(){
    struct Queue* queue = createQueue();
    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);
    enqueue(queue, 50);
    display(queue);
    dequeue(queue);
    display(queue);
    dequeue(queue);
    display(queue);
    dequeue(queue);
    dequeue(queue);
    display(queue);
    return 0;}
```

Output :-

```
Enqueued 10
Enqueued 20
enqueued 30
enqueued 40
enqueued 50
Queue: 10 20 30 40 50
dequeue 10
Queue: 20 30 40 50
Queue is Full ! cannot enqueue
dequeued 20
dequeued 30
Queue: 40 50
```