ASSIGNMENT - 3

NAME :- M.uday kumar

DEPT :- AIML

SUB :- Data structures

CODE :- CSA0389

REGNO :- 192325073

Illustrate the queue operation using following function calls of size=5 Enque (25), enqueue (37) ,Enqueue(90), Dequeue() Enqueue (15) ,Enqueue (40), Enqueue (12), Dequeue(), Dequeue(), Dequeue(); Dequeue().

Sol:- To illustrate the Queue operations for queue of size 5 with given sequence of function calls, let through each step:

Initial Queue State:

* The Queue is empty initially
* Maximum size of the 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 Queue
* Queue [90, 15, 40, 12]'
* Front = 2, Rear = 5

9. Dequeue ()

* 90 is removed from Queue
* Queue : [15, 40, 12]'
* Front = 3, Rear = 5

10. Dequeue():

* 15 is removed from Queue
* Queue [40, 12]'
* Front = 4, Rear = 5

11. Dequeue():

* 40 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

```
void enqueue( struct queue * queue,int value){
if(is full (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 Quee* queue) {
 if (is empty (queue)) {
     printf ("queue is emlty! cannot dequeue \n");
 }else{
  printf(" Dequeued %d\n", queue-> items[queue
                                        ->front]);
  queue-> front ++;
  }
}
void display (struct queue * queue) {
if (is empty (queue)) {
    printf ("queue is emlty!\n");
  }else {
    printf (" que:");
 for (int i = queue-> front ; i<= queue-> rear; i++){
  print f ("%d", queue-> items [i]);
  }
  printf (" \n");
 }
}
```

Summary of operations:

→ The operations performed show how elements are enqueued & dequed from queue.

→ The queues maximum size is never excend, & elements are dequed in order they were enqued following the first -In-First-out [FIFO] principle.

2. write a c program to implement queue operations such as ENQUEUE, DEQUE and DISPLAY

```
# include <stdio.h>
# include < stdlib.h>
# define size 5
 struct queue{
     int. items [size];
      int front;
      int rear;
 };
struct queue* create queue () {
 struct  queue * queue = (struct queue *) malloc(sizeof(struct
                                                    queue));
   queue -> front = -1;
   queue-> rear =-1;
   return queue;
 }
int is full (struct queue* queue) {
     if (queue-> rear == size-1)
          return1;
     return0;
 }
    int is empty(struct queue * queue) {
    if (queue->front ==-1 || queue->front>queue->rear)
          return 1;
    }   return0;
```

```
int main (){
  struct Queue * Queue =creat Queue();
  enquene (queue,10);
  enquene (queue,20);
  enquene (aueue,30);
  enquene( queue, 40);
  enquene(queue,50);
  display (queue);
  display (queue);
  display (queue);
  enqueue (queue),60);
  display (queue);
  dequee(queue);
  dequee (que);
  dequee (queue);
   return 0;
}
```

output :

```
Enqued 10
Enqueed 20
Enqued 30
Enqued 40
Enqued 50
Queue: 10,20,30,40,50

Dequeued 10
Queue: 20 30 40 50
Queue is full! cannot enque
Queue : 20 30 40 50
Dequeue 20
Dequeue 30
Queue : 40 50.
```