Assignment 8.

Q.N.1.

Here is program to that Implements a circular Queue.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10

// Structure for a circular queue.
typedef struct circular_queue{
int items [MAX_SIZE];
int front;
int rear;
} Circular_queue;
// initialize the queue.
void init_queue (Circular_queue *queue){
    queue -> front = 0;
    queue -> rear = MAX_SIZE -1;
    }
// Check if the queue is empty
int is_empty (circular_queue *queue){
    return (queue -> front == queue -> rear +1);
    }
// Check if the queue is full
    int is_full (circular_queue *queue){
    return (queue -> Front == 0 && queue -> rear ==
    MAX_SIZE-1) || (queue -> Front == queue ->
        rear+2); }
```

```
// insert an element into the queue

void enqueue(circular_queue * queue, int item) {
if (is_Full(queue)) {
printf("Error: queue is full \n");
return;
}

queue -> rear = (queue -> rear + 1) % MAX_SIZE;
queue -> items [queue -> rear] = item;
}

// Remove an element from the queue
int dequeue (circular_queue *queue) {
if (is_empty (queue)) {
printf ("Error: queue is empty \n");
exit(1);
}
queue -> front = (queue -> front + 1) % MAX_SIZE;
return queue -> items [queue -> front-1];
}
int main() {
circular_queue queue;
int_queue (& queue);
// insert elements into the queue
enqueue (& queue, 1);
enqueue (& queue, 2);
enqueue [& queue, 3];

//
```

```
// Remove elements into the queue.
    printf("%d\n", dequeue(&queue));
    printf("%d\n", dequeue(&queue));
    printf("%d\n", dequeue(&queue));
    return 0;
}
```

Here is an example of queue as an ADT

~~# define [queue_size])~~

```
# define QUEUE_SIZE 100
typedef struct {
    int data[QUEUE_SIZE];
    int front;
    int rear;
    int size;
} Queue;

void init(Queue *q) {
    q->front = 0;
    q->rear = QUEUE_SIZE - 1;
    q->size = 0;
}
int is_empty(Queue *q) {
    return q->size == 0;
}
int is_full(Queue *q) {
    return q->size == QUEUE_SIZE;
}
```

```
void enqueue (Queue *q, int x) {
 if (is_Full (q)) {
 printf ("Error: queue is full \n");
 return;
 }
 q->rear = (q->rear+1) % QUEUE_SIZE;
 q->data [q->rear] = x;
 q->size ++;
}
int dequeue(queue *q) {
 if (is_empty (q)) {
 printf ("Error: queue is empty \n");
 return -1;
 }
 int x = q->data [q->front];
 q->front = (q->front +1) % QUEUE_SIZE;
 q->size --;
 return x;
}
```

The primitive operations are.

1) enqueue: The operation adds an element to the rear of the queue.
2) dequeue: This operation removes an element from the front of the queue.
3) peek: This operations returns the element at the front of the queue without removing it.
4) 'is-empty': This operation returns a non-zero value if the queue is empty, and 0 if is queueue is not empty
5) 'is-full': This operation returns a non-zero value if the queue is full, and 0 if the queue is not full

Q.N2)

=) A priority queue is data structure that stores a collection of items and allows for efficient retrieval and removal of the items with the highest priority. The priority of each item is determined by a priority value associated with the item, with higher priority items being ~~rete~~ retrieved before lower priority items.

A priority queue is a type of queue that orders elements in a particular order such as highest to lowest priority. This allows elements with higher priority to be deqved before elements. The main advantage of priority queue over a a linear queue is that it allows for more efficient processing of elements. It allows elements to be added & and removed in a way that maintains the order of the queue. overall the main advantage of a priority queue is that it allows for more efficient processing of elements by prioritizing the order in which they are deqved.

In a priority queue, elements are stored in such a way that the element with the highest priority (also call the "front" element) is always at the front of the queue. The process of inserting an element into a priority queue is as follows:

1 Creat a new node with the given element and priority
2 If the priority queue is empty, set the new node as the front of the queue.
3 otherwise, compare the priority of the new node with front element. If the new node has a higher priority, set it as Front element.
4 If the new node has a lower priority, find the appropriate position for it in the queue based on its priority and insert it there.

The process of deleting an element from a priority queue is as Follows.

1) If the priority queue is empty, return null
2) otherwise, remove the front element from the queue and return it.
3) If the queue is not empty, set the next element in the queue as the new front element.

3) What is queue

⇒ We define a queue to be list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. ~~The element~~

Advantages of Circular queue over linear queue.

- Flexible in insertion & deletion: In a circular queue, elements can be added easily if the queue isn't full. But in the linear queue, elements can't be added furthermore, once the rear-end point to the last index.

- Memory efficiency: Circular queue is memory more efficient than a linear queue as we can add elements until complete. Thus no space is left over. While in linear queue once the queue is full, if we start to dequeue, the front indexes become vacant and then they can never be filled. Thus there is a wastage of space.

4) Write C functions to insert and delete an item in circular and linear queue.

⇒
```
void enqueue (int value, int * queue, int * front,
int * rear, int capacity) {
if ((*rear+1) % capacity == *front) {
// Queue is Full
   printf ("Error: queue is full \n");
   return;
    }
  * rear = (*rear+1) % capacity;
   queue [*rear] = value;
    }
```

This is function takes in the value to be inserted

Here is a C function for deleting an element from a circular queue.

```
int deque (int * queue, int *front, int *rear,
int capacity) {
 if (*front == *rear) {
// Queue is empty
   printf ("Error: queue is empty. \n");
   return -1;
  }
* front = (*front +1) % capacity;
  return queue [*front];
   }
```

Here is a C function for inserting an element into linear queue:

```
void enque (int value, int * queue, int *rear,
  int capacity) {
  if (*rear== capacity-1) {
  // queue is full
    printf (" Error: queue is Full \n");
    return;
    }
  *rear = *rear+1;
  queue [*rear] = value;
  }
```

Here is a C function for deleting an element from a linear queue

```
int dequeue (int * queue, int *front, int *rear) {
  if (*front >= *rear) {
    // queue is empty
  printf (" Error: Queue is empty \n");
    return -1;
  }
  *front = *front+1;
    return queue [*front-1];
    }
```