Array Implementation of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from the front. If we simply increment front and rear indices, then there may be problems, the front may reach the end of the array. The solution to this problem is to increase front and rear in circular manner.

Steps for enqueue:

1. Check the queue is full or not
2. If full, print overflow and exit
3. If queue is not full, increment tail and add the element

Steps for dequeue:

1. Check queue is empty or not
2. if empty, print underflow and exit
3. if not empty, print element at the head and increment head

Below is a program to implement above operation on queue

```
// C program for array implementation of queue
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// A structure to represent a queue
struct Queue {
    int front, rear, size;
    unsigned capacity;
    int* array;
};

// function to create a queue
// of given capacity.
```

```c
// It initializes size of queue as 0
struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue = (struct Queue*)malloc(
        sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;

    // This is important, see the enqueue
    queue->rear = capacity - 1;
    queue->array = (int*)malloc(
        queue->capacity * sizeof(int));
    return queue;
}

// Queue is full when size becomes
// equal to the capacity
int isFull(struct Queue* queue)
{
    return (queue->size == queue->capacity);
}

// Queue is empty when size is 0
int isEmpty(struct Queue* queue)
{
    return (queue->size == 0);
}

// Function to add an item to the queue.
// It changes rear and size
void enqueue(struct Queue* queue, int item)
{
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1)
                  % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}

// Function to remove an item from queue.
// It changes front and size
```

```
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1)
                   % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

// Function to get front of queue
int front(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
}

// Function to get rear of queue
int rear(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->rear];
}

// Driver program to test above functions./
int main()
{
    struct Queue* queue = createQueue(1000);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue\n\n",
           dequeue(queue));

    printf("Front item is %d\n", front(queue));
    printf("Rear item is %d\n", rear(queue));
```

```
    return 0;
}
```

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```