Queue | Set 2 (Linked List Implementation)

In the previous post, we introduced Queue and discussed array implementation. In this post, linked list implementation is discussed. The following two main operations must be implemented efficiently.

In a Queue data structure, we maintain two pointers, *front* and *rear*. The *front* points the first item of queue and *rear* points to last item.

enQueue() This operation adds a new node after *rear* and moves *rear* to the next node.

deQueue() This operation removes the front node and moves *front* to the next node.

```
// A C program to demonstrate linked list based implementation of queue
#include <stdlib.h>
#include <stdio.h>

// A linked list (LL) node to store a queue entry
struct QNode
{
    int key;
    struct QNode *next;
};

// The queue, front stores the front node of LL and rear stores ths
// last node of LL
struct Queue
{
    struct QNode *front, *rear;
};

// A utility function to create a new linked list node.
struct QNode* newNode(int k)
{
    struct QNode *temp = (struct QNode*)malloc(sizeof(struct QNode));
    temp->key = k;
    temp->next = NULL;
    return temp;
}

// A utility function to create an empty queue
struct Queue *createQueue()
{
    struct Queue *q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = q->rear = NULL;
    return q;
}

// The function to add a key k to q
void enQueue(struct Queue *q, int k)
{
    // Create a new LL node
    struct QNode *temp = newNode(k);

    // If queue is empty, then new node is front and rear both
```

```
    if (q->rear == NULL)
    {
       q->front = q->rear = temp;
       return;
    }

    // Add the new node at the end of queue and change rear
    q->rear->next = temp;
    q->rear = temp;
}

// Function to remove a key from given queue q
struct QNode *deQueue(struct Queue *q)
{
    // If queue is empty, return NULL.
    if (q->front == NULL)
       return NULL;

    // Store previous front and move front one node ahead
    struct QNode *temp = q->front;
    q->front = q->front->next;

    // If front becomes NULL, then change rear also as NULL
    if (q->front == NULL)
       q->rear = NULL;
    return temp;
}

// Driver Program to test anove functions
int main()
{
    struct Queue *q = createQueue();
    enQueue(q, 10);
    enQueue(q, 20);
    deQueue(q);
    deQueue(q);
    enQueue(q, 30);
    enQueue(q, 40);
    enQueue(q, 50);
    struct QNode *n = deQueue(q);
    if (n != NULL)
      printf("Dequeued item is %d", n->key);
    return 0;
}
```

Run on IDE

Output:

```
Dequeued item is 30
```

Time Complexity: Time complexity of both operations enqueue() and dequeue() is O(1) as we only change few pointers in both operations. There is no loop in any of the operations.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.

Category: Linked List Queue