

Linked List implementation of Queue

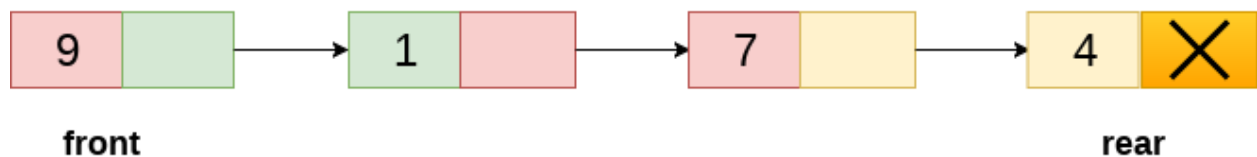
Due to the drawbacks discussed in the previous section of this tutorial, the array implementation can not be used for the large scale applications where the queues are implemented. One of the alternative of array implementation is linked list implementation of queue.

The storage requirement of linked representation of a queue with n elements is $O(n)$ while the time requirement for operations is $O(1)$.

In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.



Linked Queue

Operation on Linked Queue

here are two basic operations which can be implemented on the linked queues. The operations are Insertion and Deletion.

Insert operation

The insert operation append the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

Firstly, allocate the memory for the new node ptr by using the following statement.

Firstly, allocate the memory for the new node ptr by using the following statement.

1. `Ptr = (struct node *) malloc (sizeof(struct node));`

There can be the two scenario of inserting this new node ptr into the linked queue.

In the first scenario, we insert element into an empty queue. In this case, the condition **front = NULL** becomes true. Now, the new element will be added as the only element of the queue and the next pointer of front and rear pointer both, will point to NULL.

```
1. ptr -> data = item;
2.   if(front == NULL)
3.   {
4.       front = ptr;
5.       rear = ptr;
6.       front -> next = NULL;
7.       rear -> next = NULL;
8.   }
```

In the second case, the queue contains more than one element. The condition front = NULL becomes false. In this scenario, we need to update the end pointer rear so that the next pointer of rear will point to the new node ptr. Since, this is a linked queue, hence we also need to make the rear pointer point to the newly added node **ptr**. We also need to make the next pointer of rear point to NULL.

```
1. rear -> next = ptr;
2.   rear = ptr;
3.   rear->next = NULL;
```

In this way, the element is inserted into the queue.

Deletion

Deletion operation removes the element that is first inserted among all the queue elements.

Firstly, we need to check either the list is empty or not. The condition front == NULL becomes true if the list is empty, in this case , we simply write underflow on the console and make exit.

Otherwise, we will delete the element that is pointed by the pointer front. For this purpose, copy the node pointed by the front pointer into the pointer ptr. Now, shift the front pointer, point to its next node and free the node pointed by the node ptr. This is done by using the following statements.

```
1. ptr = front;
2.   front = front -> next;
3.   free(ptr);
```

The algorithm and C function is given as follows.

