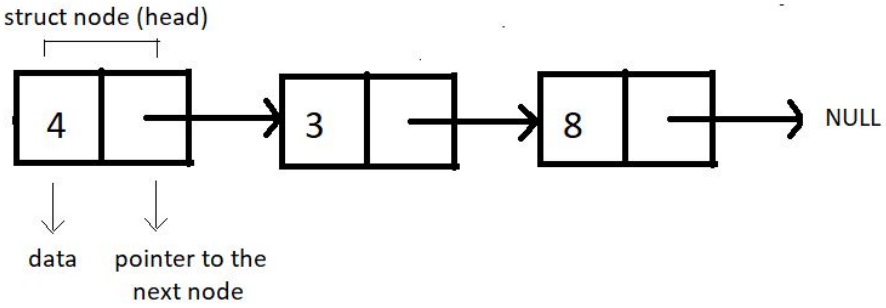
Queue Using Linked Lists

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-45

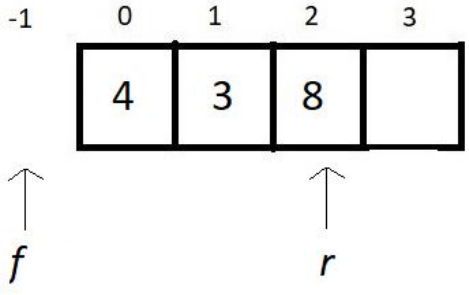
Up until now, queues had been implemented using arrays. We have another alternative that is a must to learn and has been examiners' favorite topic, implementing queues using linked lists. Today, we'll see how to implement queues using linked lists and some of its operations.

Implementing queues using linked lists tests your proficiency in using/ handling both queues and linked lists. And, in case if you have missed either or both of these, I would recommend visiting them first before proceeding.

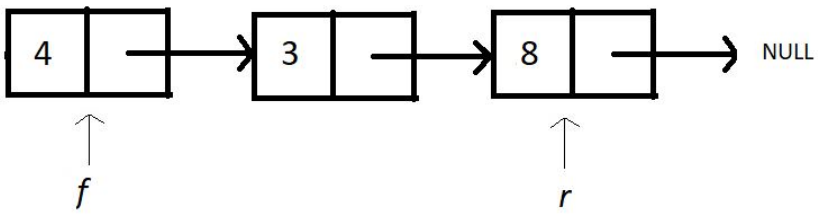
If you remember, linked lists are chain-like structures with nodes having two parts, an integer variable to hold data and another node pointer to hold the address of the next node. Below illustrated is a linked list with three nodes. The last node points to NULL. And the first node is called the *head*.



Moving to the basics of a queue, a queue represents a line or sequence of elements where the elements follow the FIFO discipline. The element inserting the first gets removed the first. We maintained two index variables, f, and r, to mark the beginning and the end of the queue. Below illustrated is a queue with three elements.



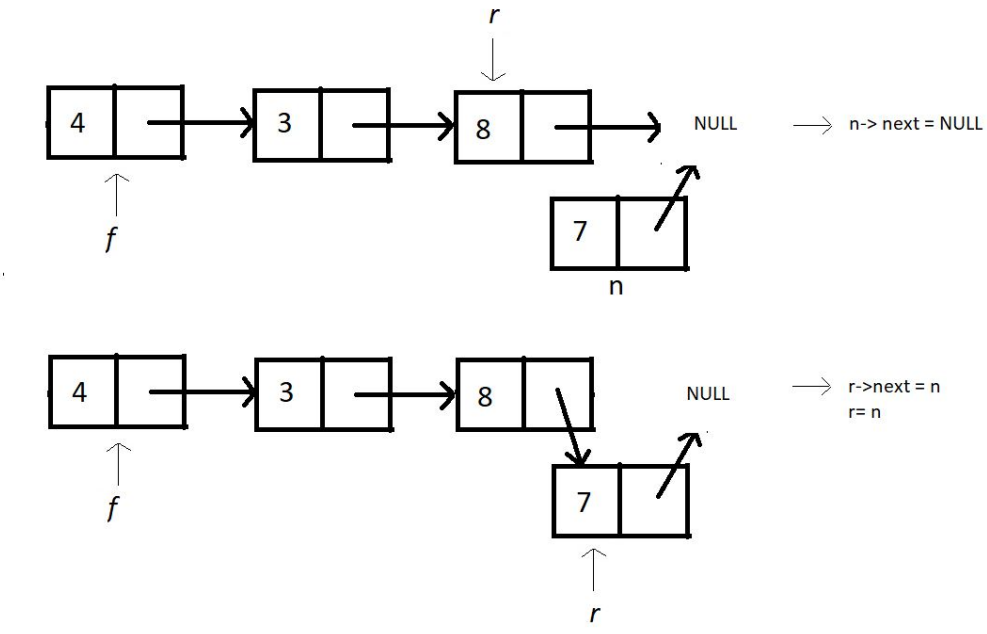
Since we are implementing this queue using a linked list, the index variables are no longer integers. These become the pointers to the front and the rear nodes. And the queue somewhat starts looking like this.



Enqueue in a queue linked list:

Enqueuing in a queue linked list is very much similar to just inserting at the end in a linked list. As we discussed this thoroughly in our past lectures, you should not find this difficult. Inserting a new node at the end requires you to follow few steps:

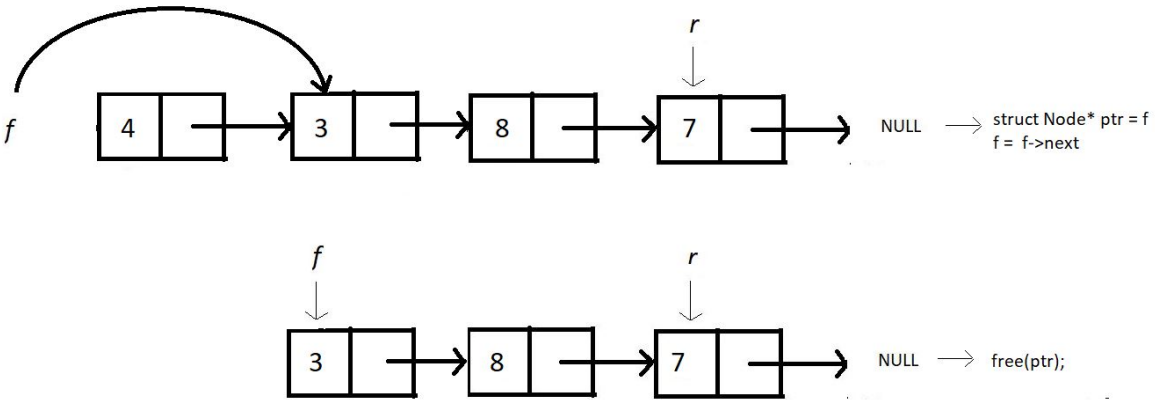
1. Check if there is a space left in the heap for a new node.
2. If there is, create a new node *n, assign* it memory in heap, and fill its data with the new value the user has given.
3. Point the next member of this new node *n* to NULL, and point the next member of the *r* to *n*. And make *r* equal to *n*. And we are done.
4. There is one exception here. When we insert the first element, both *f and r* are pointing to NULL. So, instead of just making *r* equal to *n*, we make *f* equal to *n* as well. This marks the beginning of the list.



Dequeue in a queue linked list:

Dequeuing in a queue linked list is very much similar to deleting the head node in a linked list. Deleting the head node from the list requires you to follow few steps:

1. Check if the queue list is already not empty using the *isEmpty* function.
2. If it is, return -1. Else create a new node *ptr* and make it equal to the *f* node. And don't forget to store the data of the *f* node in some integer variable.
3. Make the *f* equal to the next member of *f*, and free the node *ptr*. Return the value you stored.



Condition for *isEmpty*:

The only condition for the queue linked list to be empty is that the *f* node is NULL, which means there is no beginning, hence no element.

Condition for *isFull*:

Queues implemented using linked lists never usually become full until the space in the heap memory is exhausted. Therefore, the only condition for the queue linked list to be full is that the *new* node becomes NULL when created.

We could very easily finish implementing queues using linked lists because we learned it extensively in our previous lectures. We started from the basics of both queues and linked lists and could complete everything. I expect you all to write the codes for all these operations yourself, and do tell me in whichever way if you could write the program on your own. I'll be more than happy to see you all progress. We will anyways discuss the program in C in the next lecture.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll program everything discussed today in C. Till then, keep coding.