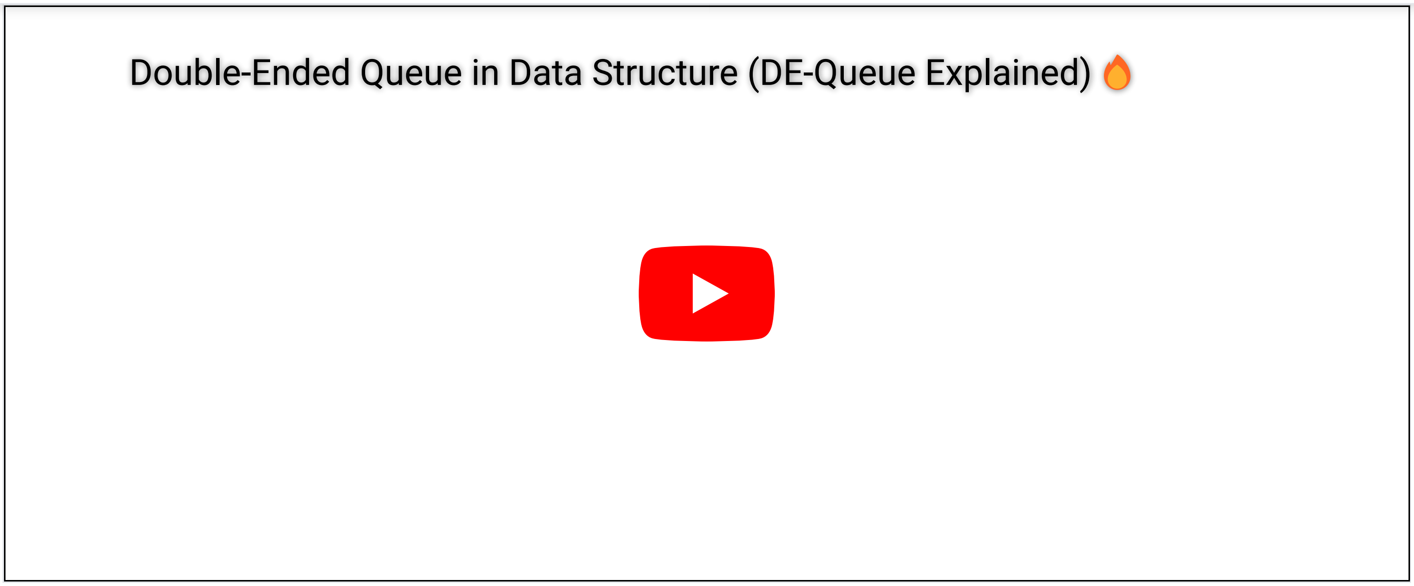
CodeWithHarry Menu

Login



Show Course Contents

Overview Q&A Downloads Announcements

Double-Ended Queue in Data Structure (DE-Queue Explained)

In the last lecture, we finished learning about queues. We saw both queues and circular queues. We implemented queues using both arrays and linked lists. We saw all their operations. There is actually nothing left there in queues except one interesting topic, which is DE-Queue. It should not be confused with the dequeue we learnt. It is **Double Ended Queues**.

We had certain characteristics in normal queues, which I would like to summarize here:

1. A queue is very similar to the real-life queue, where you stand in the last and wait for your turn.
2. Similarly, the elements get inserted from one end and exit from the other.
3. We had two pointers cum index variables to maintain the two ends of this queue.
4. We followed the FIFO principle throughout the lectures.

And now, in **DEQueue, we don't follow the FIFO principle**. As the name suggests, this variant of the queue is double-ended. This means that unlike normal queues where insertion could only happen at the rear end, and deletion at the front end, these double-ended queues have the freedom to insert and delete elements from the end of their choice.

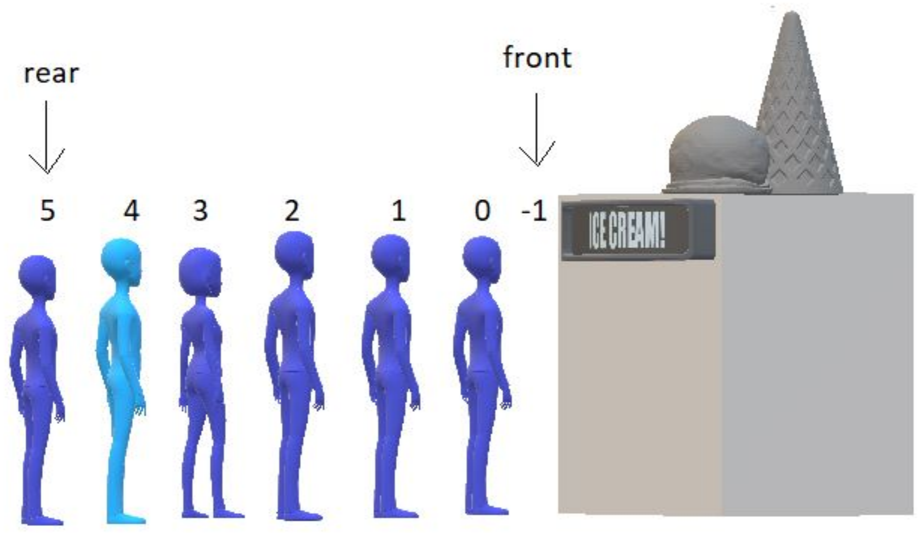
Double-ended queues, hence, have the following characteristics:

1. They don't follow the FIFO discipline.
2. Insertion can be done at both the ends of the queue.
3. Deletion can also be done from both ends of the queue.

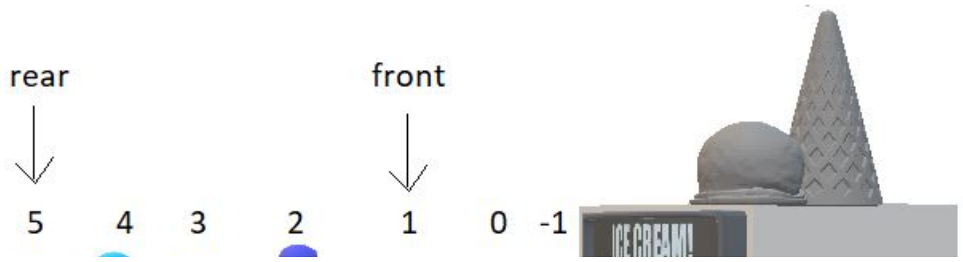
You would assume the implementation part of double-ended queues to be on the tough side, but believe me, it is straightforward to consume. I'll use illustrations to make you understand things better.

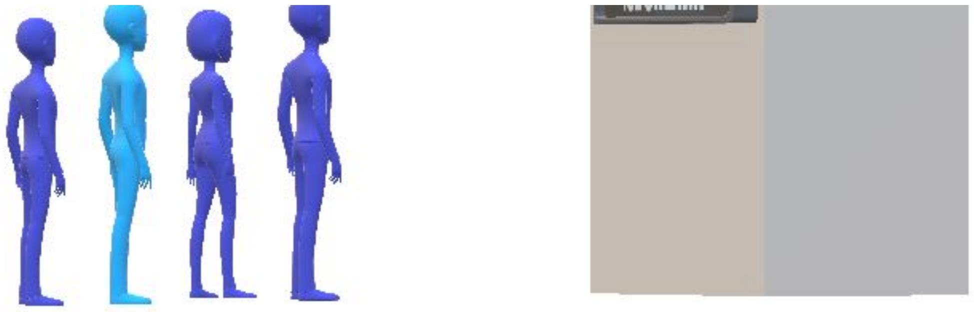
Insertion in a DEQueue:

Insertion in a DEQueue is very intuitive. Follow the illustration below:

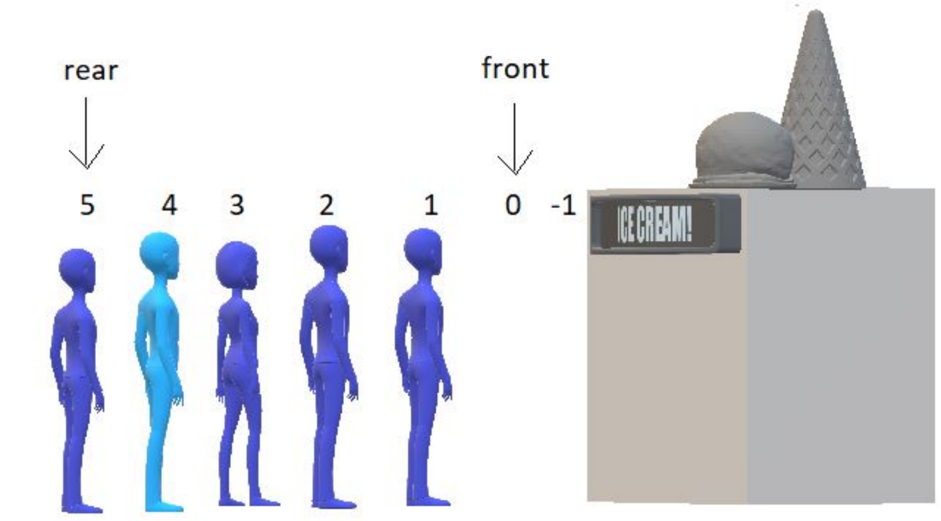


Now since the front has no space to insert, you can only insert at the rear end. But if the front manages to have some space after some dequeuing, then our condition would be something like this:



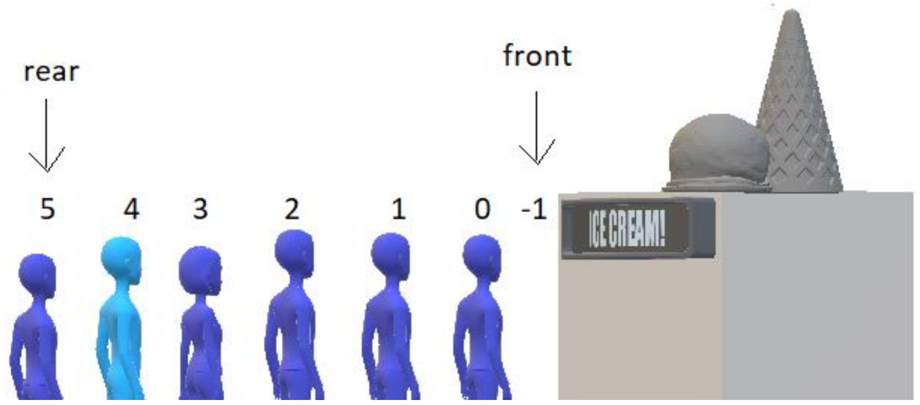


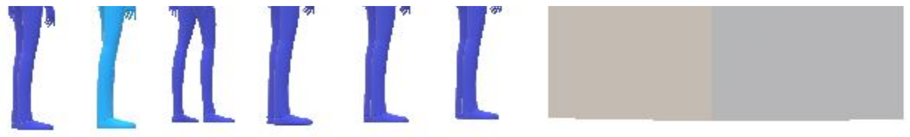
Now, we have 2 places to fill in front as well. And in DEQueue, we have no restrictions. We would just fill our new element at the front and decrease its value by 1. And that would be it. See the results below:



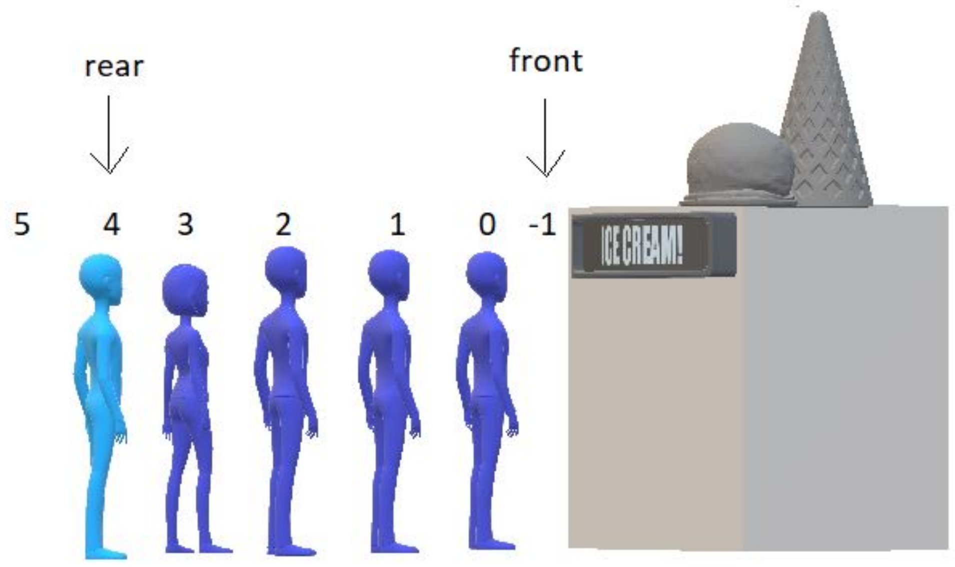
Deletion in a DEQueue:

Deletion in a DEQueue is very similar to what we did above. Follow the illustration below:





Now, for one moment, think of the rear as the front end. You would simply then increase the front value by 1 and delete the element at the new front. Similarly, here we first delete the element at rear and decrease the value of the rear by 1. See the results below.



And yeah, we are done deleting the element from the rear end. And inserting at the front end. Moving further, DEQueues are of two types:

1. Restricted Input DEQueue
2. Restricted Output DEQueue

Restricted Input DEQueue:

Input restricted DEQueues don’t allow insertion on the front end. But you can delete from both ends.

Restricted Output DEQueue:

Output restricted DEQueues don’t allow deletion from the rear end. But you can perform the insertion on both the ends.

Now the main part is that you would write the program for implementing the

Double Ended Queue ADT yourself this time! I know you are capable of doing that. For your convenience, I would like to discuss the ADT part. I would mention all the functionalities one would expect in DEQueues. So, yeah, let's see the DEQueue ADT.

DEQueue ADT:

The data part would be the same as the queue. I wouldn't repeat things. Refer to the Queue ADT from here.

Methods:

All the operations except the enqueue and dequeue will remain the same as that of the queue. In place of enqueue and dequeue, we would have:

1. enqueueF()
2. enqueueR()
3. dequeueF()
4. dequeueR()

You can even have more of these, as initialise(), print(), etc. This was our abstract data type, DEQueue. Do implement their programs, and had it not been under your capability, I would have taught it for sure. But you can surely do that on your own. Do tell me if you could. That would be one of those reasons for me to be happy.

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 start with **sorting algorithms**. Till then, keep coding.

Previous Next