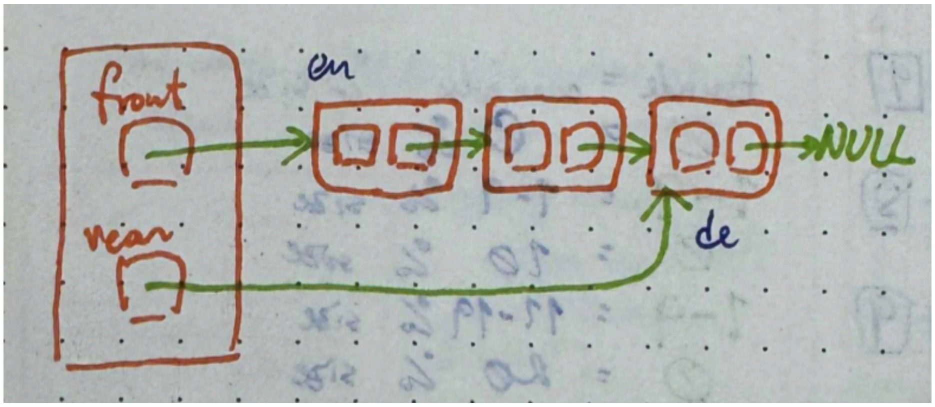
LL Queue

Variation 1

Queue is a linked list with **front** and **rear**. FRONT is a pointer to the **first** node of the list, REAR is a pointer to the **last** node of the list.



Operations	**Checklist**	**Example**
`Queue* initialize();`	☐ Allocate memory for the queue structure ☐ Initialize both front and rear pointers to NULL ☐ Return the pointer to the new queue	`Queue* Q = initialize();`
`bool isFull(Queue* q);`	☐ Return false (linked list can never be full)	
`bool isEmpty(Queue* q);`	☐ Queue is empty if the front pointer of the queue is NULL	

`void enqueue(Queue* q, int value);`	☐ Allocate memory for a new node
	☐ Set the data of the new node and set its next pointer to NULL
	☐ Check if the queue is currently empty
	☐ If the queue is empty, the new node is both the front and the rear
	☐ If the queue is not empty, link the current rear to the new node
	☐ Then, update the rear pointer to the new node
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`enqueue(Q, 4);`
		
		`After:`
		`  front -> 1 -> 3 -> 5 -> 4 -> NULL`
		`  rear -------------------^`

`int dequeue(Queue* q);`	☐ Check if the queue is empty before attempting to dequeue
	☐ Store a temporary pointer to the front node
	☐ Store the data of the front node
	☐ Move the front pointer to the next node
	☐ If the queue becomes empty after this operation, update the rear pointer to NULL
	☐ Free the memory of the old front node
	☐ Return the stored value
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`int value = dequeue(Q);`
		
		`After:`
		`  front -> 3 -> 5 -> NULL`
		`  rear ---------^`

`int front(Queue* q);`	☐ Check if the queue is empty ☐ Otherwise, return the data of the front	

Operations	Checklist	Example
`Queue* initialize();`	☐ Allocate memory for the queue structure ☐ Initialize both front and rear pointers to NULL ☐ Return the pointer to the new queue	`Queue* Q = initialize();`
`bool isFull(Queue* q);`	☐ Return false (linked list can never be full)	
`bool isEmpty(Queue* q);`	☐ Queue is empty if the front pointer of the queue is NULL	

`void enqueue(Queue* q, int value);`	☐ Allocate memory for a new node
	☐ Set the data of the new node and set its next pointer to NULL
	☐ Check if the queue is currently empty
	☐ If the queue is empty, the new node is both the front and the rear
	☐ If the queue is not empty, link the current rear to the new node
	☐ Then, update the rear pointer to the new node
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`enqueue(Q, 4);`
		
		`After:`
		`  front -> 1 -> 3 -> 5 -> 4 -> NULL`
		`  rear -------------------^`

`int dequeue(Queue* q);`	☐ Check if the queue is empty before attempting to dequeue
	☐ Store a temporary pointer to the front node
	☐ Store the data of the front node
	☐ Move the front pointer to the next node
	☐ If the queue becomes empty after this operation, update the rear pointer to NULL
	☐ Free the memory of the old front node
	☐ Return the stored value
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`int value = dequeue(Q);`
		
		`After:`
		`  front -> 3 -> 5 -> NULL`
		`  rear ---------^`

	node	
`void display(Queue* q);`	☐ Check if the queue is empty	

s	Checklist	Example
`nitialize();`	□ Allocate memory for the queue structure □ Initialize both front and rear pointers to NULL □ Return the pointer to the new queue	`Queue* Q = initialize();`
`ull(Queue* q);`	□ Return false (linked list can never be full)	
`mpty(Queue* q);`	□ Queue is empty if the front pointer of the queue is NULL	

`ueue(Queue* q, int`	□ Allocate memory for a new node
	□ Set the data of the new node and set its next pointer to NULL
	□ Check if the queue is currently empty
	□ If the queue is empty, the new node is both the front and the rear
	□ If the queue is not empty, link the current rear to the new node
	□ Then, update the rear pointer to the new node
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`enqueue(Q, 4);`
		
		`After:`
		`  front -> 1 -> 3 -> 5 -> 4 ->`
		`  rear -------------------^`

`eue(Queue* q);`	□ Check if the queue is empty before attempting to dequeue
	□ Store a temporary pointer to the front node
	□ Store the data of the front node
	□ Move the front pointer to the next node
	□ If the queue becomes empty after this operation, update the rear pointer to NULL
	□ Free the memory of the old front node
	□ Return the stored value
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`int value = dequeue(Q);`
		
		`After:`
		`  front -> 3 -> 5 -> NULL`
		`  rear ---------^`

	□ Create a temporary pointer to traverse the queue	

s	Checklist	Example
`nitialize();`	□ Allocate memory for the queue structure □ Initialize both front and rear pointers to NULL □ Return the pointer to the new queue	`Queue* Q = initialize();`
`ull(Queue* q);`	□ Return false (linked list can never be full)	
`mpty(Queue* q);`	□ Queue is empty if the front pointer of the queue is NULL	

`ueue(Queue* q, int`	□ Allocate memory for a new node
	□ Set the data of the new node and set its next pointer to NULL
	□ Check if the queue is currently empty
	□ If the queue is empty, the new node is both the front and the rear
	□ If the queue is not empty, link the current rear to the new node
	□ Then, update the rear pointer to the new node
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`enqueue(Q, 4);`
		
		`After:`
		`  front -> 1 -> 3 -> 5 -> 4 ->`
		`  rear -------------------^`

`eue(Queue* q);`	□ Check if the queue is empty before attempting to dequeue
	□ Store a temporary pointer to the front node
	□ Store the data of the front node
	□ Move the front pointer to the next node
	□ If the queue becomes empty after this operation, update the rear pointer to NULL
	□ Free the memory of the old front node
	□ Return the stored value
		`Before:`
		`  front -> 1 -> 3 -> 5 -> NULL`
		`  rear --------------^`
		
		`int value = dequeue(Q);`
		
		`After:`
		`  front -> 3 -> 5 -> NULL`
		`  rear ---------^`

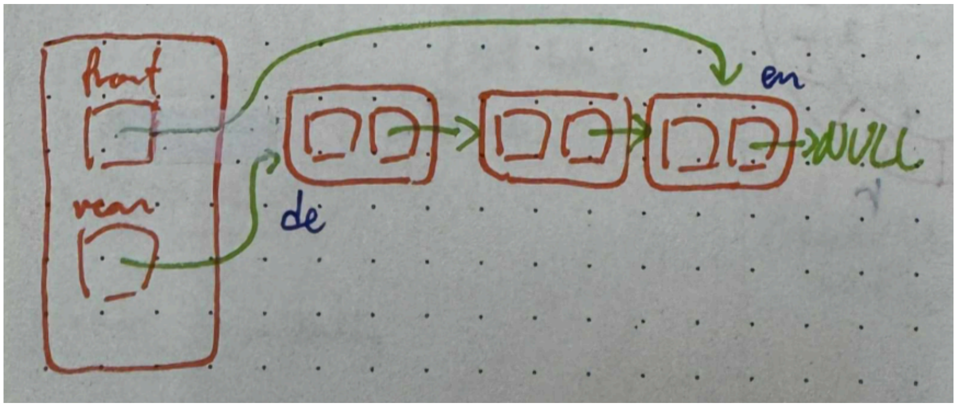
	□ Loop through the list until the end (NULL) is reached	

Note:

For most operations, it is also common to return a **boolean value** representing whether the operation is successful or not.

Variation 2

Queue is a linked list with **front** and **rear**. FRONT is a pointer to the **last** node of the list, REAR is a pointer to the **first** node of the list.


front
rear
en
de
NULL