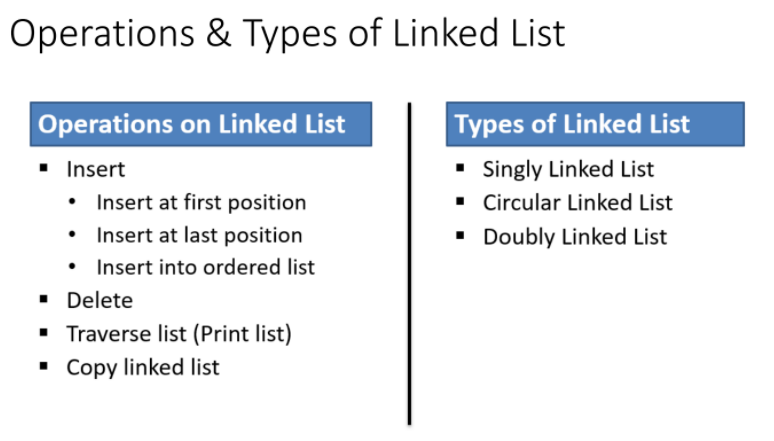
* Define Linked List. How is it differ from Array Data Structure?  
    
  A linked list is a non-sequential collection of data items.

A Linked List is a Linear Data Structure.

* + A LL is the collection of nodes that are randomly stored.
  + Each node is divided into two parts, the first part represents the data of the element and the second part contains the address of next node.
  + The last node of the list stored null value as the address.
  + It is possible for a list to have no nodes at all, such a list is called empty list.  
      
    

### **Differences between Array and Linked list**

|  |  |
| --- | --- |
| **Array** | **Linked list** |
| An array is a collection of elements of a similar data type. | A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address. |
| Array elements store in a **contiguous memory** location. | Linked list elements can be **stored anywhere in the memory** or randomly stored. |
| Array works with a **static memory.** Here static memory means that the **memory size is fixed and cannot be changed at the run time.** | The Linked list works with **dynamic memory**. Here, dynamic memory means that the **memory size can be changed at the run time** according to our requirements. |
| Array elements are **independent of each other.** | Linked list elements are **dependent on each other**. As each node contains the address of the next node so to access the next node, we need to access its previous node. |
| Array **takes more time** while performing any operation like **insertion, deletion**, etc. | Linked list **takes less time** while performing any operation like **insertion, deletion**, etc. |
| **Accessing any element** in an array is **faster** as the element in an array can be directly accessed through the index. | **Accessing an element** in a linked list is **slower** as it starts traversing from the first element of the linked list. |
| In the case of an array, **memory is allocated at compile-time.** | In the case of a linked list, **memory is allocated at run time.** |
| Memory utilization is inefficient in the array. For example, if the size of the array is 6, and array consists of 3 elements only then the rest of the space will be unused. | Memory utilization is efficient in the case of a linked list as the memory can be allocated or deallocated at the run time according to our requirement. |

**What is the limitation of Simple Queue? Which are the ways to overcome limitations of Simple Queue? Explain with suitable example**

The main limitation of queues is one of the basic operations of deleting an element from it is cumbersome.

When we add an element in Queue, the rear pointer is increased by 1 whereas, when we remove an element front pointer is increased by 1.

But an array implementation of queue this may cause the problem as follows:  
  
Consider operations performed on a Queue (with SIZE = 5) as follows:

1. Initially empty Queue is there so, front = 0 and rear = -1

2. When we add 5 elements to queue, the state of the queue becomes as follows with front = 0 and rear = 4

10

20

30

40

50

3. Now suppose we delete 2 elements from Queue then, the state of the Queue becomes as follows, with front = 2 and rear = 4

30

40

50

4. Now, actually we have deleted 2 elements from the queue so, there should be space for another 2 elements in the queue, but as the rear pointer is pointing at last position and Queue overflow condition  
(Rear == SIZE-1) is true, we can’t insert the new element in the queue even if it has an empty space.  
To overcome this problem there is another variation of queue called [CIRCULAR QUEUE](http://letusc-sharp.blogspot.com/2012/10/circular-queue.html)

What are the advantages of Doubly Linked List over Singly Linked List?

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

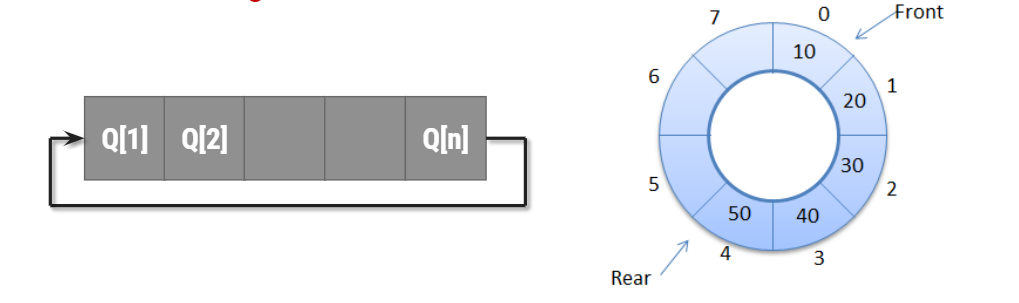
3) We can quickly insert a new node before a given node.

Explain insertion operation in the Circular queue with all the conditions.

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle

In circular queue the last node is connected back to the first node to make a circle.

It is also called as “Ring buffer”.



Operations on Circular Queue:

Front: Get the front item from queue.

Rear: Get the last item from queue.

enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).

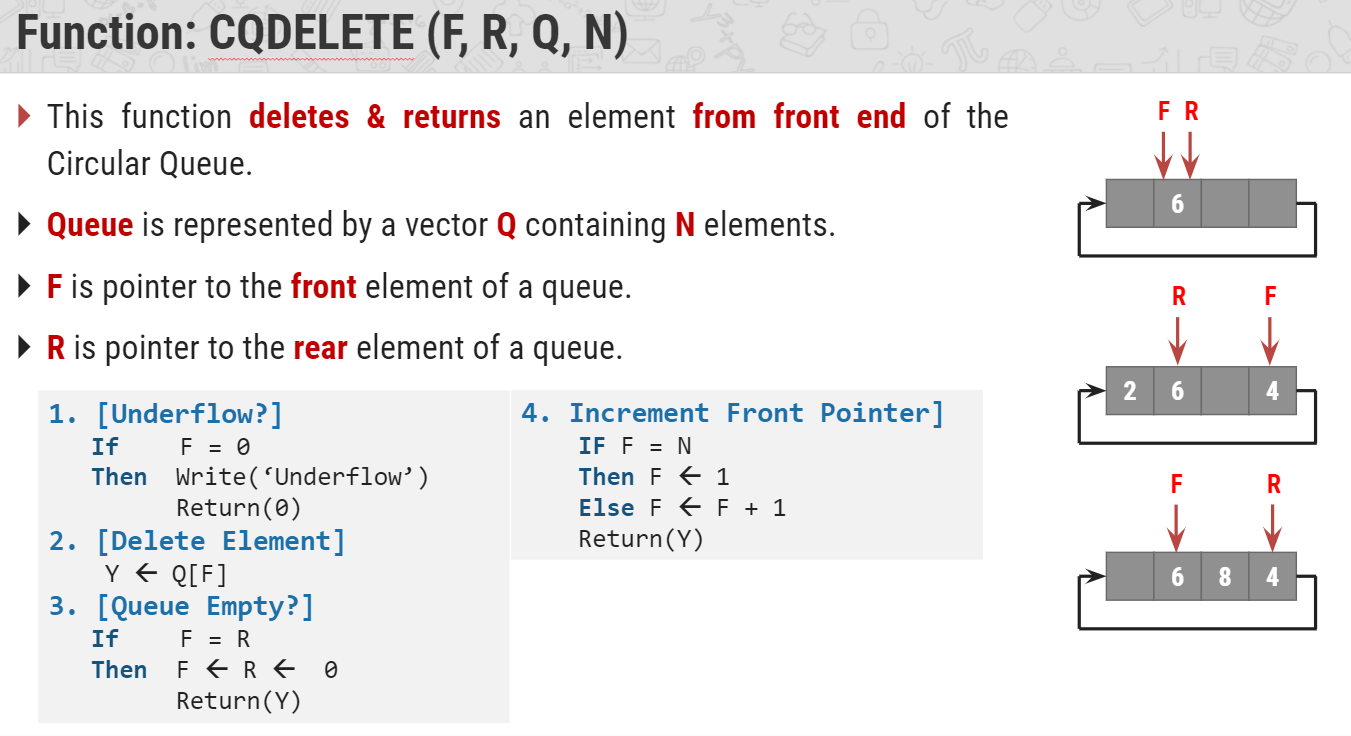
If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

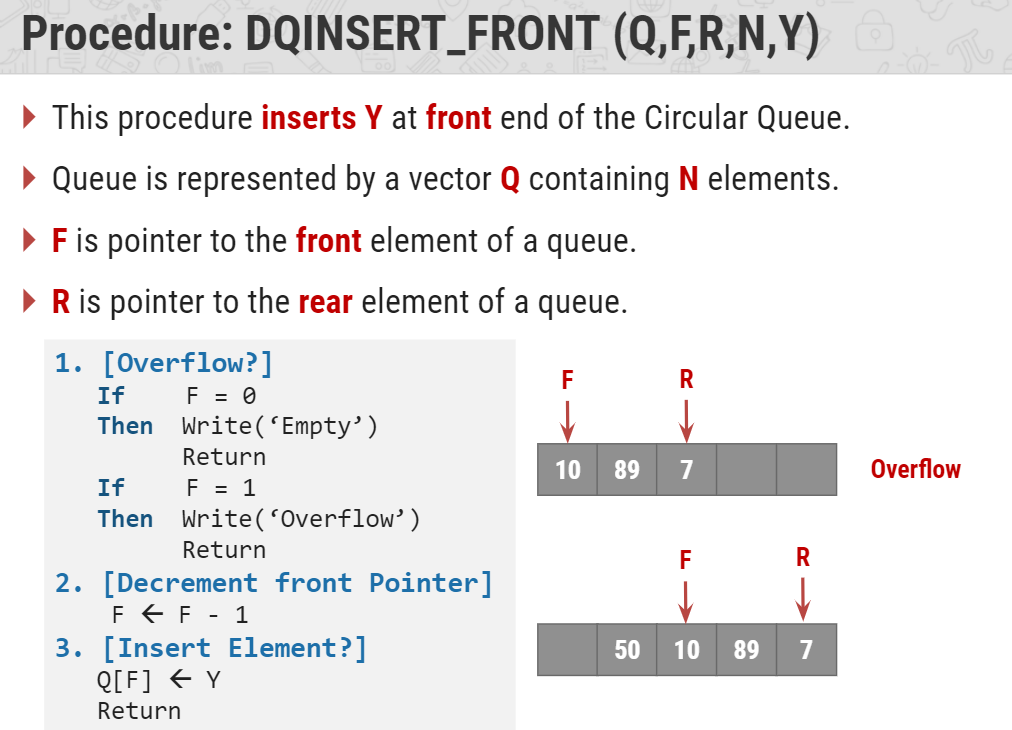
deQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

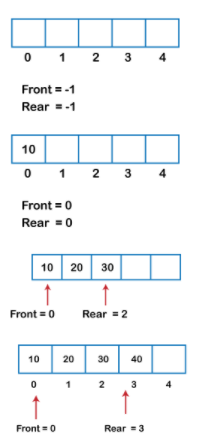
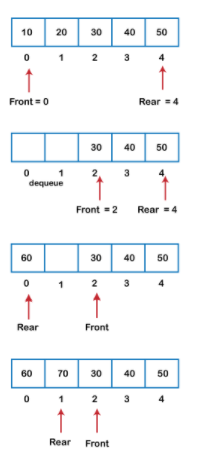
Check whether queue is Empty means check (front==-1).

If it is empty then display Queue is empty. If queue is not empty then step 3

Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

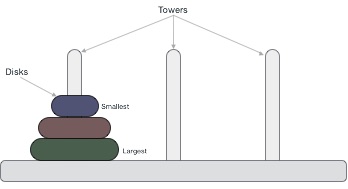






Discuss tower of hanoi problem for N=3 discs with recursive tracing.

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −



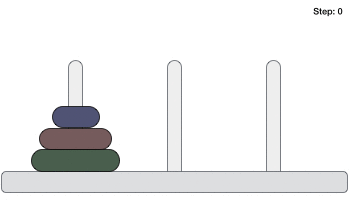
These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

## Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

* Only one disk can be moved among the towers at any given time.
* Only the "top" disk can be removed.
* No large disk can sit over a small disk.

Following is an animated representation of solving a Tower of Hanoi puzzle with three disks.



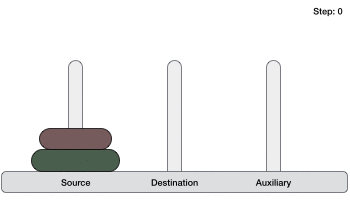
Tower of Hanoi puzzle with n disks can be solved in minimum **2n−1** steps. This presentation shows that a puzzle with 3 disks has taken **23 - 1 = 7** steps.

## Algorithm

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, **source**, **destination** and **aux** (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks −

* First, we move the smaller (top) disk to aux peg.
* Then, we move the larger (bottom) disk to destination peg.
* And finally, we move the smaller disk from aux to destination peg.



So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk **n** from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.

The steps to follow are −

**Step 1** − Move n-1 disks from **source** to **aux**

**Step 2** − Move nth disk from **source** to **dest**

**Step 3** − Move n-1 disks from **aux** to **dest**

A recursive algorithm for Tower of Hanoi can be driven as follows −

START

Procedure Hanoi(disk, source, dest, aux)

IF disk == 1, THEN

move disk from source to dest

ELSE

Hanoi(disk - 1, source, aux, dest) // Step 1

move disk from source to dest // Step 2

Hanoi(disk - 1, aux, dest, source) // Step 3

END IF

END Procedure

STOP

What is data structure? Explain types of data structure with an example.

<https://www.javatpoint.com/data-structure-introduction#:~:text=Data%20Structure%20can%20be%20defined,%2C%20Stack%2C%20Queue%2C%20etc>.

Write an algorithm or C function for push and pop operation of stack.

## Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

* **Step 1** − Checks if the stack is full.
* **Step 2** − If the stack is full, produces an error and exit.
* **Step 3** − If the stack is not full, increments **top** to point next empty space.
* **Step 4** − Adds data element to the stack location, where top is pointing.
* **Step 5** − Returns success.



If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.

### **Algorithm for PUSH Operation**

A simple algorithm for Push operation can be derived as follows −

begin procedure push: stack, data

if stack is full

return null

endif

top ← top + 1

stack[top] ← data

end procedure

Implementation of this algorithm in C, is very easy. See the following code −

**Example**

void push(int data) {

if(!isFull()) {

top = top + 1;

stack[top] = data;

} else {

printf("Could not insert data, Stack is full.\n");

}

}

## Pop Operation

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead **top** is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

A Pop operation may involve the following steps −

* **Step 1** − Checks if the stack is empty.
* **Step 2** − If the stack is empty, produces an error and exit.
* **Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.
* **Step 4** − Decreases the value of top by 1.
* **Step 5** − Returns success.



### **Algorithm for Pop Operation**

A simple algorithm for Pop operation can be derived as follows −

begin procedure pop: stack

if stack is empty

return null

endif

data ← stack[top]

top ← top - 1

return data

end procedure

Implementation of this algorithm in C, is as follows −

**Example**

int pop(int data) {

if(!isempty()) {

data = stack[top];

top = top - 1;

return data;

} else {

printf("Could not retrieve data, Stack is empty.\n");

}

}

Write C function or an algorithm for inserting and deleting an element into simple queue.

<https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm>

Write an algorithm for doubly linked list that performs following operations:(i) Insert a node after the node whose address is M (ii) Delete the node whose address is OLD

<https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm>