Queues

1. **Queue**

**Send Feedback**

A queue is a ?

1. FIFO(First In First Out) answer
2. LIFO(Last In First Out)
3. Ordered Array
4. Linear Tree
5. **Correct order**

**Send Feedback**

If the elements “p”, “q”, “r” and “s” are placed in a queue and are deleted one at a time, in what order will they be removed?

1. Srqp
2. Pqrs answer
3. Srpq
4. qrsp
5. **Queue Using LL**

**Send Feedback**

#### Implement a Queue Data Structure specifically to store integer data using a Singly Linked List.

#### The data members should be private.

#### You need to implement the following public functions :

##### 1. Constructor:

#### It initialises the data members as required.

##### 2. enqueue(data) :

#### This function should take one argument of type integer. It enqueues the element into the queue and returns nothing.

##### 3. dequeue() :

#### It dequeues/removes the element from the front of the queue and in turn, returns the element being dequeued or removed. In case the queue is empty, it returns -1.

##### 4. front() :

#### It returns the element being kept at the front of the queue. In case the queue is empty, it returns -1.

##### 5. getSize() :

#### It returns the size of the queue at any given instance of time.

##### 6. isEmpty() :

#### It returns a boolean value indicating whether the queue is empty or not.

##### Operations Performed on the Stack:

Query-1(Denoted by an integer 1): Enqueues an integer data to the queue.

Query-2(Denoted by an integer 2): Dequeues the data kept at the front of the queue and returns it to the caller.

Query-3(Denoted by an integer 3): Fetches and returns the data being kept at the front of the queue but doesn't remove it, unlike the dequeue function.

Query-4(Denoted by an integer 4): Returns the current size of the queue.

Query-5(Denoted by an integer 5): Returns a boolean value denoting whether the queue is empty or not.

##### Input Format:

The first line contains an integer 'q' which denotes the number of queries to be run against each operation on the queue.

Then the test cases follow.

Every 'q' lines represent an operation that needs to be performed.

For the enqueue operation, the input line will contain two integers separated by a single space, representing the type of the operation in integer and the integer data being enqueued into the queue.

For the rest of the operations on the queue, the input line will contain only one integer value, representing the query being performed on the queue.

##### Output Format:

For Query-1, you do not need to return anything.

For Query-2, prints the data being dequeued from the queue.

For Query-3, prints the data kept on the front of the queue.

For Query-4, prints the current size of the queue.

For Query-5, prints 'true' or 'false'(without quotes).

Output for every query will be printed in a separate line.

##### Note:

You are not required to print anything explicitly. It has already been taken care of. Just implement the functions.

##### Constraints:

1 <= q <= 10^5

1 <= x <= 5

-2^31 <= data <= 2^31 - 1 and data != -1

Where 'q' is the total number of queries being performed on the queue, 'x' is the range for every query and data represents the integer pushed into the queue.

Time Limit: 1 second

##### Sample Input 1:

7

1 17

1 23

1 11

2

2

2

2

##### Sample Output 1:

17

23

11

-1

##### Sample Input 2:

3

2

1 10

4

##### Sample Output 2:

-1

1

1. **Insert Element**

**Send Feedback**

In optimized linked list implementation of a queue, where does a new element be inserted?

1. At the head of linked list
2. At the tail of the linked list answer
3. At the centre position in the linked list
4. None
5. **Worst Case**

**Send Feedback**

In linked list implementation of queue, if only front pointer is maintained, which of the following operation take worst case time (i.e. O(n) )?

1. Insertion from rear answer
2. Deletion from front
3. Both
4. None
5. **Stack Using 2 Queues**

**Send Feedback**

#### Implement a Stack Data Structure specifically to store integer data using two Queues.

#### There should be two data members, both being Queues to store the data internally. You may use the inbuilt Queue.

#### Implement the following public functions :

##### 1. Constructor:

#### It initialises the data members as required.

##### 2. push(data) :

#### This function should take one argument of type integer. It pushes the element into the stack and returns nothing.

##### 3. pop() :

#### It pops the element from the top of the stack and in turn, returns the element being popped or deleted. In case the stack is empty, it returns -1.

##### 4. top :

#### It returns the element being kept at the top of the stack. In case the stack is empty, it returns -1.

##### 5. size() :

#### It returns the size of the stack at any given instance of time.

##### 6. isEmpty() :

#### It returns a boolean value indicating whether the stack is empty or not.

##### Operations Performed on the Stack:

Query-1(Denoted by an integer 1): Pushes an integer data to the stack.

Query-2(Denoted by an integer 2): Pops the data kept at the top of the stack and returns it to the caller.

Query-3(Denoted by an integer 3): Fetches and returns the data being kept at the top of the stack but doesn't remove it, unlike the pop function.

Query-4(Denoted by an integer 4): Returns the current size of the stack.

Query-5(Denoted by an integer 5): Returns a boolean value denoting whether the stack is empty or not.

##### Input Format:

The first line contains an integer 'q' which denotes the number of queries to be run against each operation in the stack.

Then the test cases follow.

Every 'q' lines represent an operation that needs to be performed.

For the push operation, the input line will contain two integers separated by a single space, representing the type of the operation in integer and the integer data being pushed into the stack.

For the rest of the operations on the stack, the input line will contain only one integer value, representing the query being performed on the stack.

##### Output Format:

For Query-1, you do not need to return anything.

For Query-2, prints the data being popped from the stack.

For Query-3, prints the data kept on the top of the stack.

For Query-4, prints the current size of the stack.

For Query-5, prints 'true' or 'false'(without quotes).

Output for every query will be printed in a separate line.

##### Note:

You are not required to print anything explicitly. It has already been taken care of. Just implement the function.

##### Constraints:

1 <= q <= 100

1 <= x <= 5

-2^31 <= data <= 2^31 - 1 and data != -1

Where 'q' is the total number of queries being performed on the stack, 'x' is the range for every query and data represents the integer pushed into the stack.

Time Limit: 1 second

##### Sample Input 1:

6

1 13

1 47

4

5

2

3

##### Sample Output 1:

2

false

47

13

##### Sample Input 2:

4

5

2

1 10

5

##### Sample Output 2:

true

-1

false

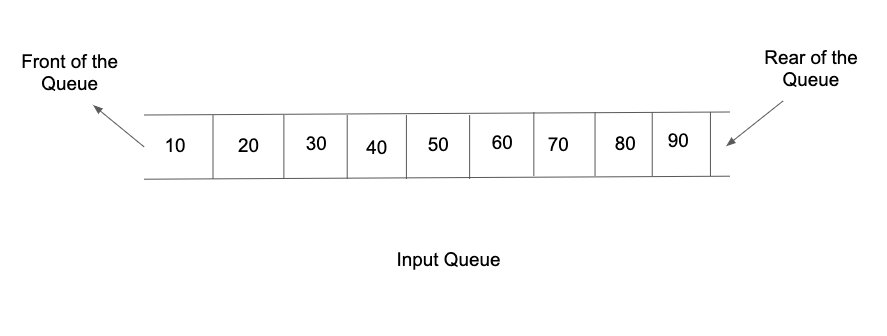
Assignment

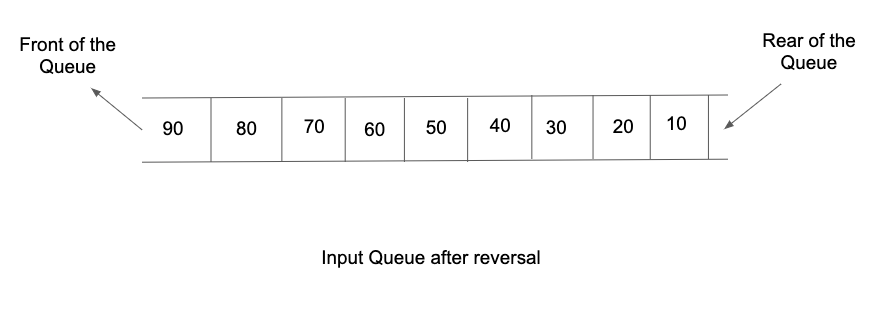
**1.Reverse Queue**

**Send Feedback**

#### You have been given a queue that can store integers as the data. You are required to write a function that reverses the populated queue itself without using any other data structures.

##### Example:





##### Input Format:

The first list of input contains an integer 't' denoting the number of test cases/queries to be run.

Then the test cases follow.

The first line input for each test case/query contains an integer N, denoting the total number of elements in the queue.

The second line of input contains N integers separated by a single space, representing the order in which the elements are enqueued into the queue.

##### Output Format:

For each test case/query, the only line of output prints the order in which the queue elements are dequeued, all of them separated by a single space.

Output for every test case/query will be printed on a new line.

##### Note:

You are not required to print the expected output explicitly, it has already been taken care of. Just make the changes in the input queue itself.

##### Constraints:

1 <= t <= 100

1 <= N <= 10^4

-2^31 <= data <= 2^31 - 1

Time Limit: 1sec

##### Sample Input 1:

1

6

1 2 3 4 5 10

##### Note:

Here, 1 is at the front and 10 is at the rear of the queue.

##### Sample Output 1:

10 5 4 3 2 1

##### Sample Input 2:

2

5

2 8 15 1 10

3

10 20 30

##### Sample Output 2:

10 1 15 8 2

30 20 10

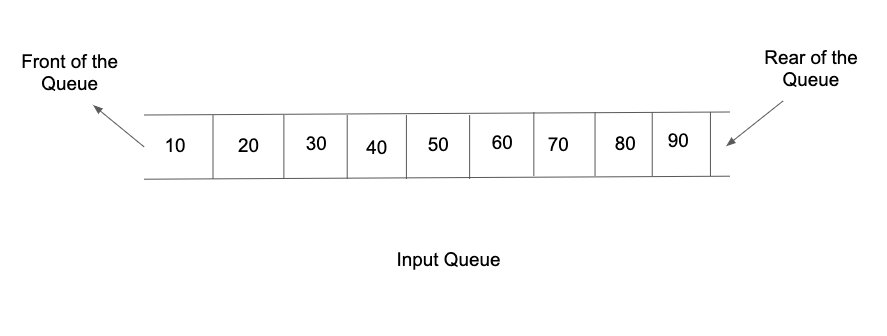
**2. Reverse the First K Elements in the Queue**

**Send Feedback**

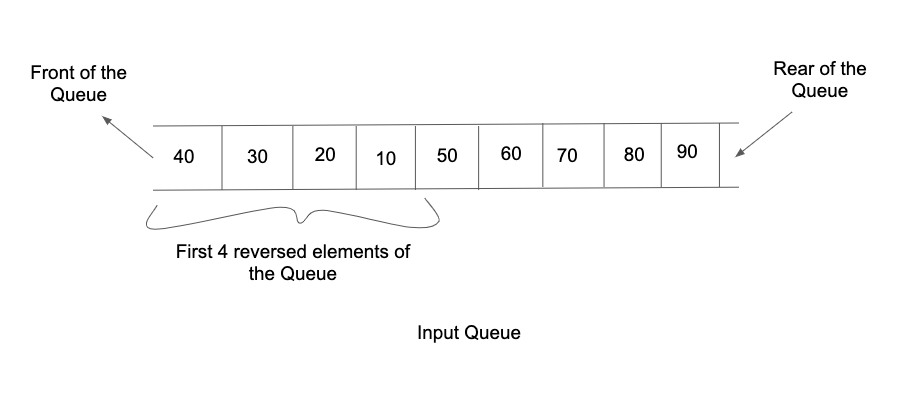
#### For a given queue containing all integer data, reverse the first K elements.

#### You have been required to make the desired change in the input queue itself.

##### Example:



For the above input queue, if K = 4 then after reversing the first 4 elements, the queue will be updated as:



##### Input Format :

The first line of input would contain two integers N and K, separated by a single space. They denote the total number of elements in the queue and the count with which the elements need to be reversed respectively.

The second line of input contains N integers separated by a single space, representing the order in which the elements are enqueued into the queue.

##### Output Format:

The only line of output prints the updated order in which the queue elements are dequeued, all of them separated by a single space.

##### Note:

You are not required to print the expected output explicitly, it has already been taken care of. Just make the changes in the input queue itself.

##### Contraints :

1 <= N <= 10^6

1 <= K <= N

-2^31 <= data <= 2^31 - 1

Time Limit: 1sec

##### Sample Input 1:

5 3

1 2 3 4 5

##### Sample Output 1:

3 2 1 4 5

##### Sample Input 2:

7 7

3 4 2 5 6 7 8

##### Sample Output 2:

8 7 6 5 2 4 3