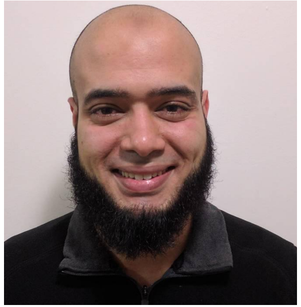
Data Structures

Circular Queue

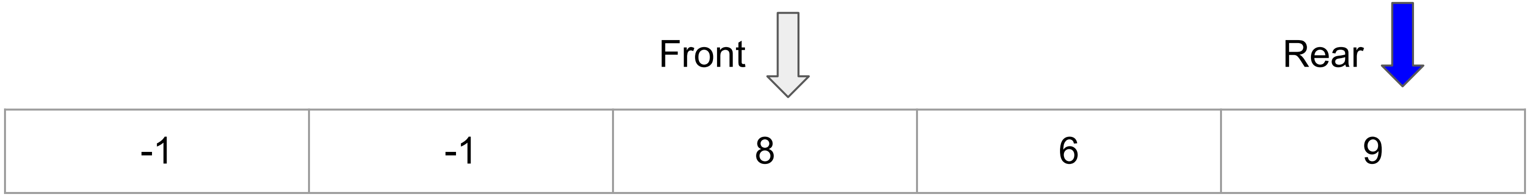
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Array-based: Front-Read approach

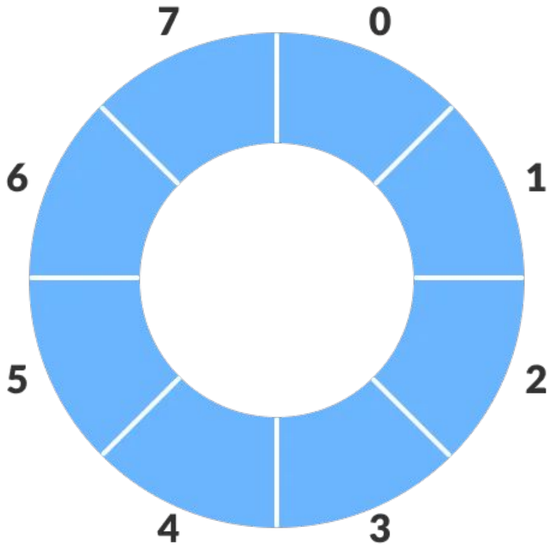
- We will have 2 indices: front and rear representing start to end in array
    - When we enqueue element we add it in rear
    - When we dequeue element we shift front to the right ⇒ O(1)
- Enqueue 3: ERROR Queue is full!
- Wait, but there are slots empty in the begin!
    - This is a critical drawback in this approach
    - How to solve? Think for 15 minutes!



-1	-1	8	6	9

Circular Queue

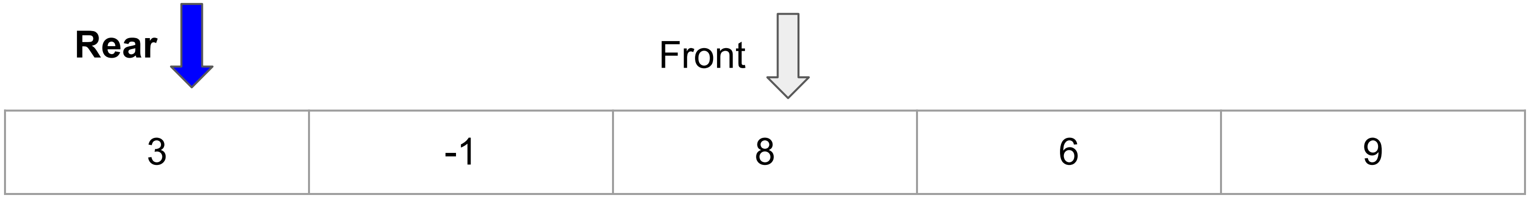
- There is a simple way to solve the previous space issue
- Simply, think in the array as a circle
  - On right, an array of 8 elements, as a circle
- That is, after the last element, there is actually another element, which is position 0
- Now, the queue is full IFF all elements are in use



Img src

Array-based Circular queue

- We will have 2 indices: front and rear representing start to end in array
  - When we enqueue element we add it in rear
  - When we dequeue element we shift font to the right ⇒ O(1)
- Enqueue 3
  - Now move from the last index to index 0 and put the new element
  - Observe: Rear now is **BEFORE** front



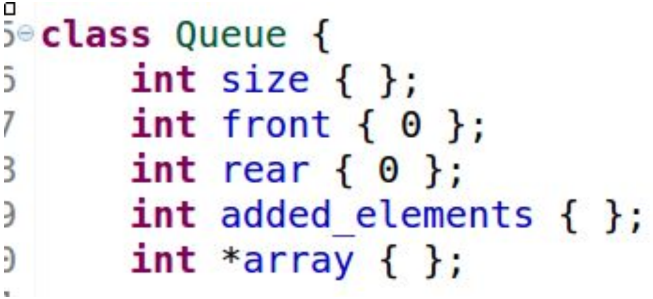
3	-1	8	6	9

Initial values for rear & front

- There are several approaches for that
  - In every approach, we have to be **consistent** in the whole implementation
  - **Careful** conditions for IsEmpty and IsFull
- Possible initializations
  - rear = front = -1 [initially equal]
  - rear = front = 0 [initially equal]
  - rear = -1 and front = 0 [initially !equal]
  - rear = size - 1 and front = 0 [initially !equal]
- **int added_elements = 0;**
  - To avoid tricky conditions and simplify coding, maintain counter for number of elements!

Circular Queue: Data Structure

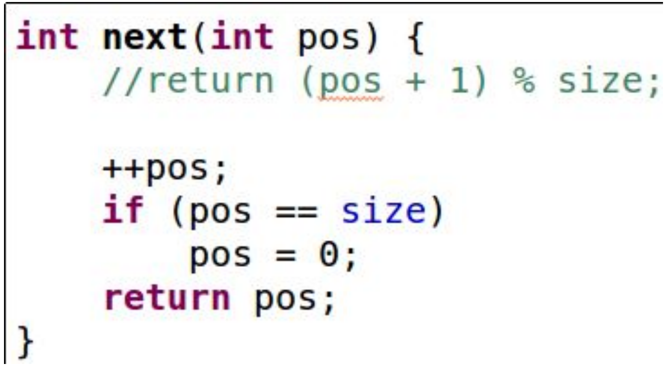
- We initially use **front = rear = 0**
  - To add a new element, **add in rear** and **move** rear
  - To dequeue element, **get front** and **move** front



```
class Queue {
	int size { };
	int front { 0 };
	int rear { 0 };
	int added_elements { };
	int *array { };
```

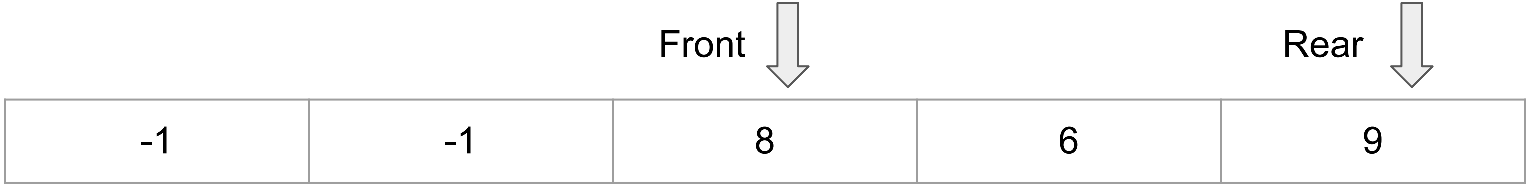
Circular Queue: Move index

- To move an index step forward consider
  - If this is **last element in the array**, next position = 0
  - We can do that with if condition (efficient)
  - Or with simple mod
- Assume size = 5. Let’s try positions from 0 to 5
  - 0 % 5 = 0
  - 1 % 5 = 1
  - 2 % 5 = 2
  - 3 % 5 = 3
  - 4 % 5 = 4
  - **5 % 5 = 0**



```
int next(int pos) {
    //return (pos + 1) % size;

    ++pos;
    if (pos == size)
        pos = 0;
    return pos;
}
```



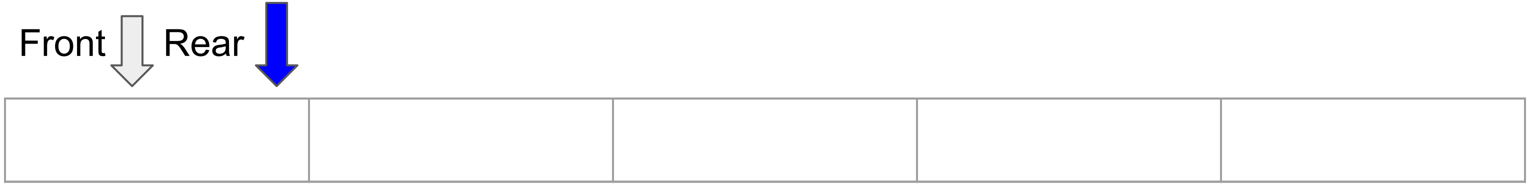
Front

Rear

-1	-1	8	6	9

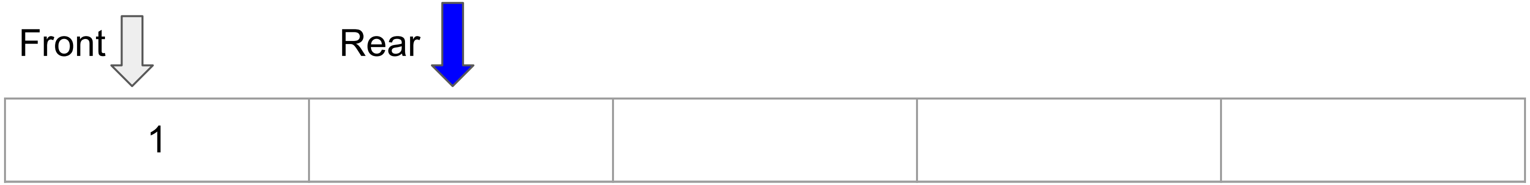
Let’s Simulate: Queue of size 5

- Initially an empty queue. Both rear = front = 0
- Observe: **Empty** queue with rear == front



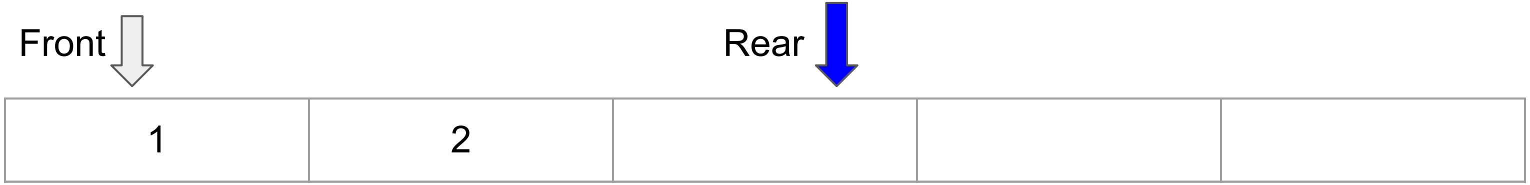
Let’s Simulate: Queue of size 5

- Enqueue (1) ⇒ Add in rear position and move it



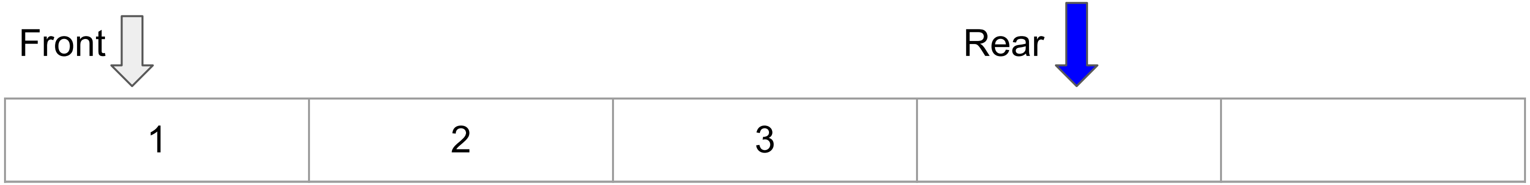
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2)



Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3)



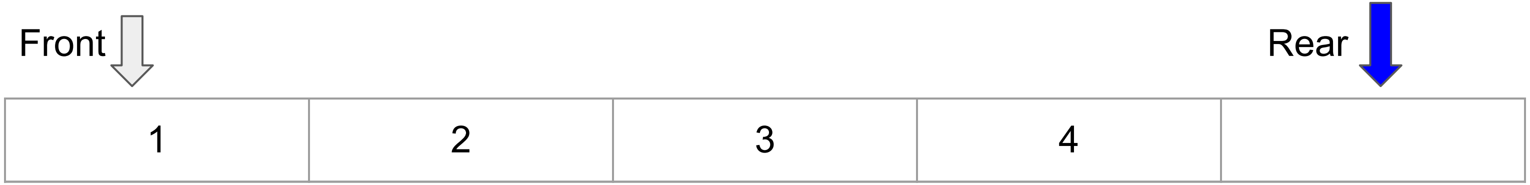
Front

Rear

1	2	3		

Let’s Simulate: Queue of size 5

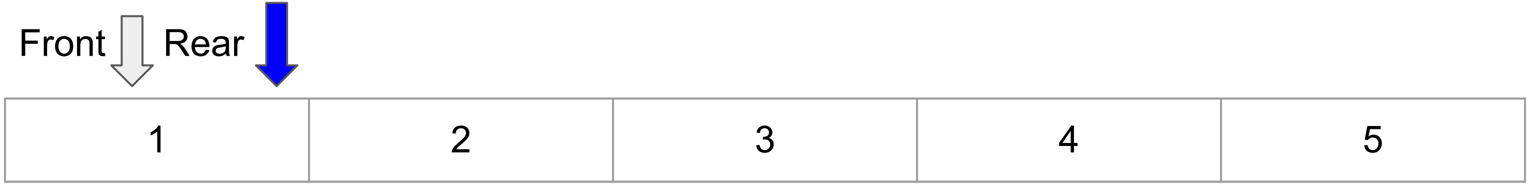
- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4)
- Observe: rear at last array position
  - One more enqueue and it moves the index to 0



1	2	3	4	

Let’s Simulate: Queue of size 5

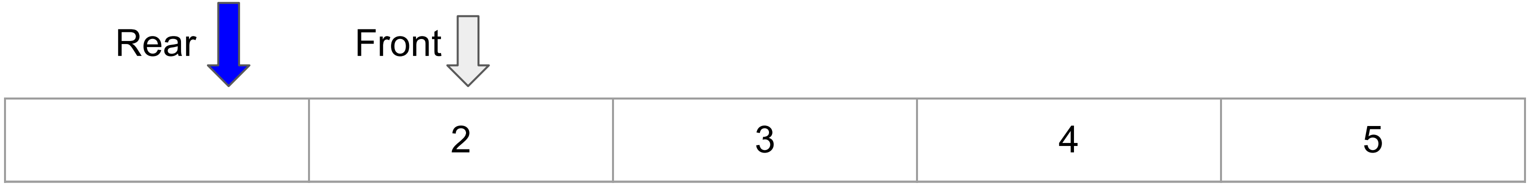
- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Observe: full queue but also rear == front
  - How can we know array is empty or full this way!
  - We **can’t!**
  - Use the added_elements variable
    - 0 = empty
    - 5 = full



1	2	3	4	5

Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Observe: Front after Rear



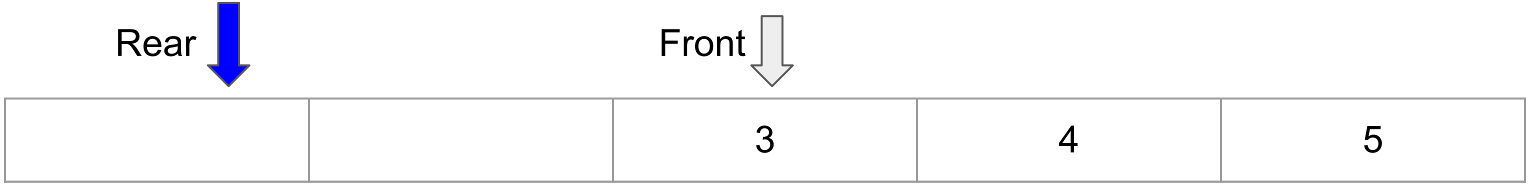
Rear

Front

	2	3	4	5

Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2



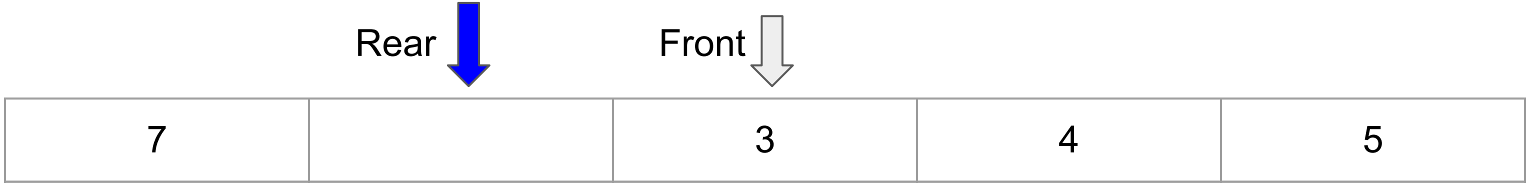
Rear

Front

		3	4	5

Let’s Simulate: Queue of size 5

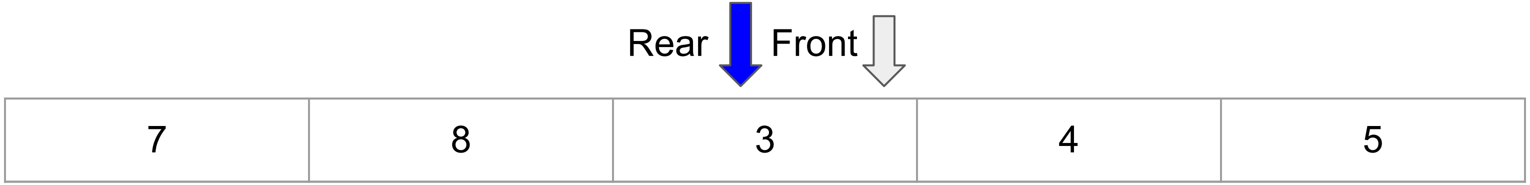
- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7)



	Rear	Front		
7		3	4	5

Let’s Simulate: Queue of size 5

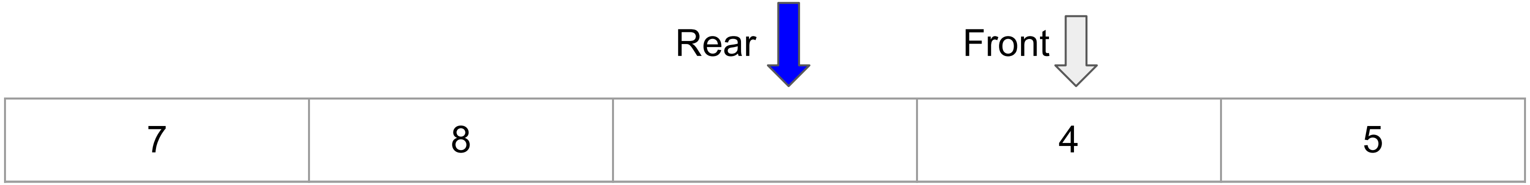
- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Again full but rear = front = index 2



7	8	3	4	5

Let’s Simulate: Queue of size 5

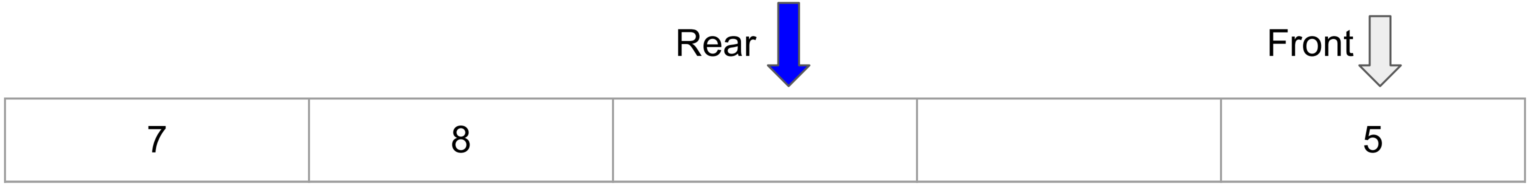
- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3



7	8		4	5

Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3
- Dequeue ⇒ 4



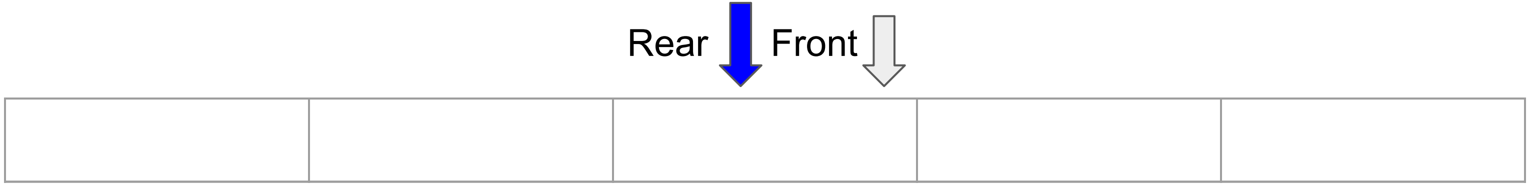
Rear

Front

7	8			5

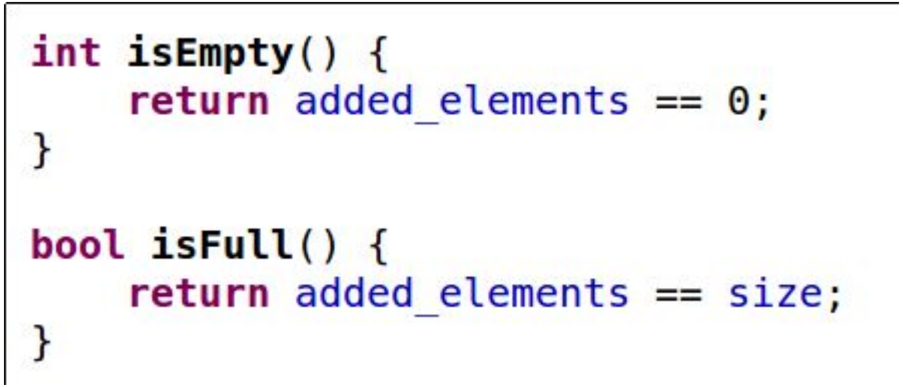
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3
- Dequeue ⇒ 4
- Dequeue, Dequeue, Dequeue ⇒ 5, 7, 8
    - Observe: empty with front = rear = 2



IsEmpty? IsFull?

- Trivially handled using added_elements

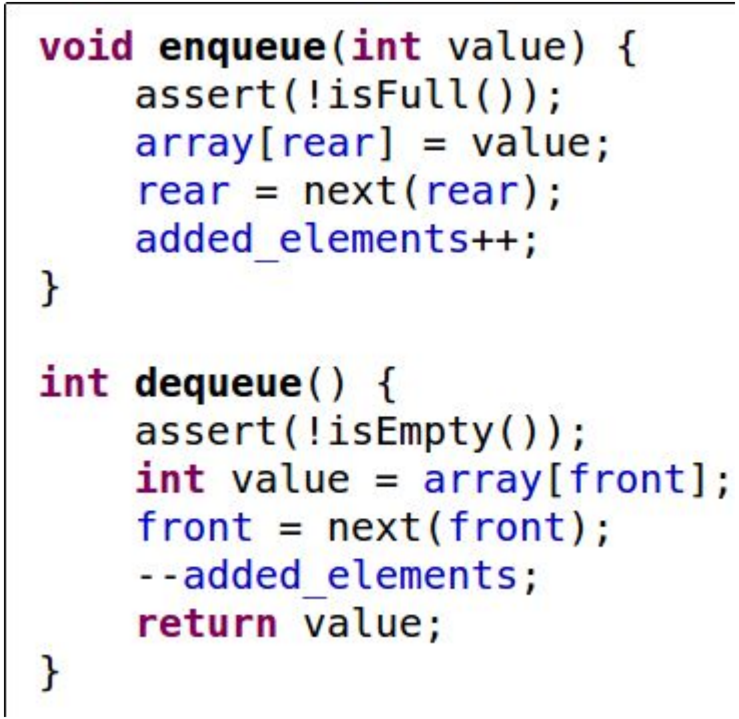


```
int isEmpty() {
    return added_elements == 0;
}

bool isFull() {
    return added_elements == size;
}
```

Enqueue and Dequeue

- Enqueue: Add in rear and move
- Dequeue: Get from front and move
- Direct!

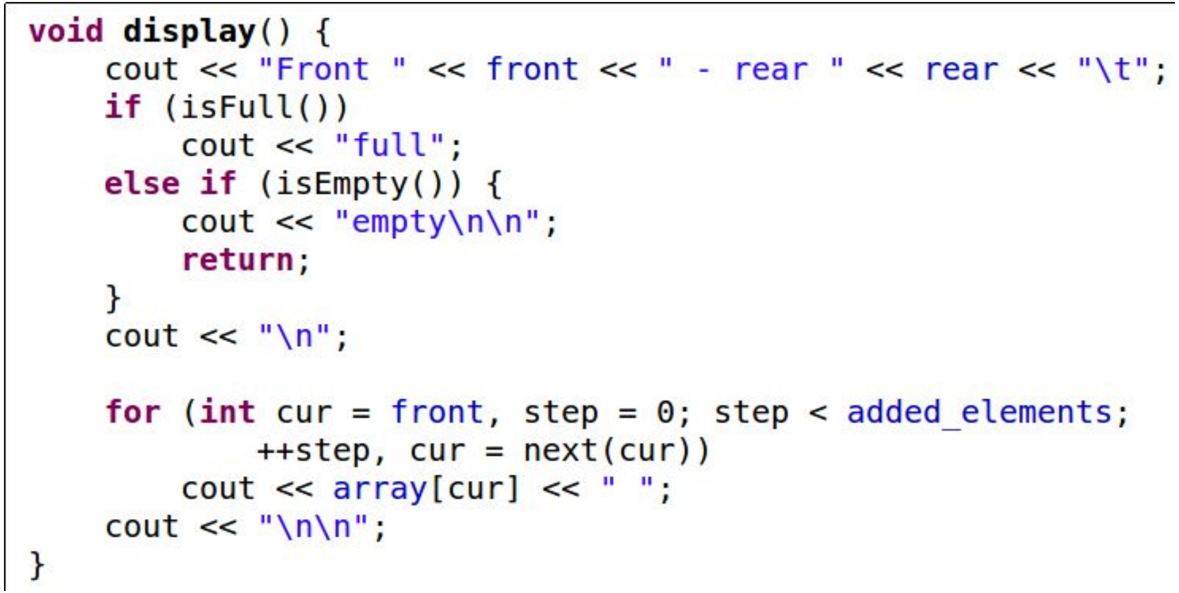


```
void enqueue(int value) {
    assert(!isFull());
    array[rear] = value;
    rear = next(rear);
    added_elements++;
}

int dequeue() {
    assert(!isEmpty());
    int value = array[front];
    front = next(front);
    --added_elements;
    return value;
}
```

Display Queue

- Simply start from the front and count based on added_elements



```cpp
void display() {
    cout << "Front " << front << " - rear " << rear << "\t";
    if (isFull())
        cout << "full";
    else if (isEmpty()) {
        cout << "empty\n\n";
        return;
    }
    cout << "\n";

    for (int cur = front, step = 0; step < added_elements;
            ++step, cur = next(cur))
        cout << array[cur] << " ";
    cout << "\n\n";
}
```

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”