Algorithm
and
Data Structures

Algorithm

- A “**finite sequence**” of “**well defined**” computational steps that transforms “**input**” into the “**output**”.

- Basic constructs of an algorithm.
  - Linear Sequence – statements that follow one after the other.
  - Conditional – “if then else”
  - Loop – sequence of statements that are repeated a number of times.

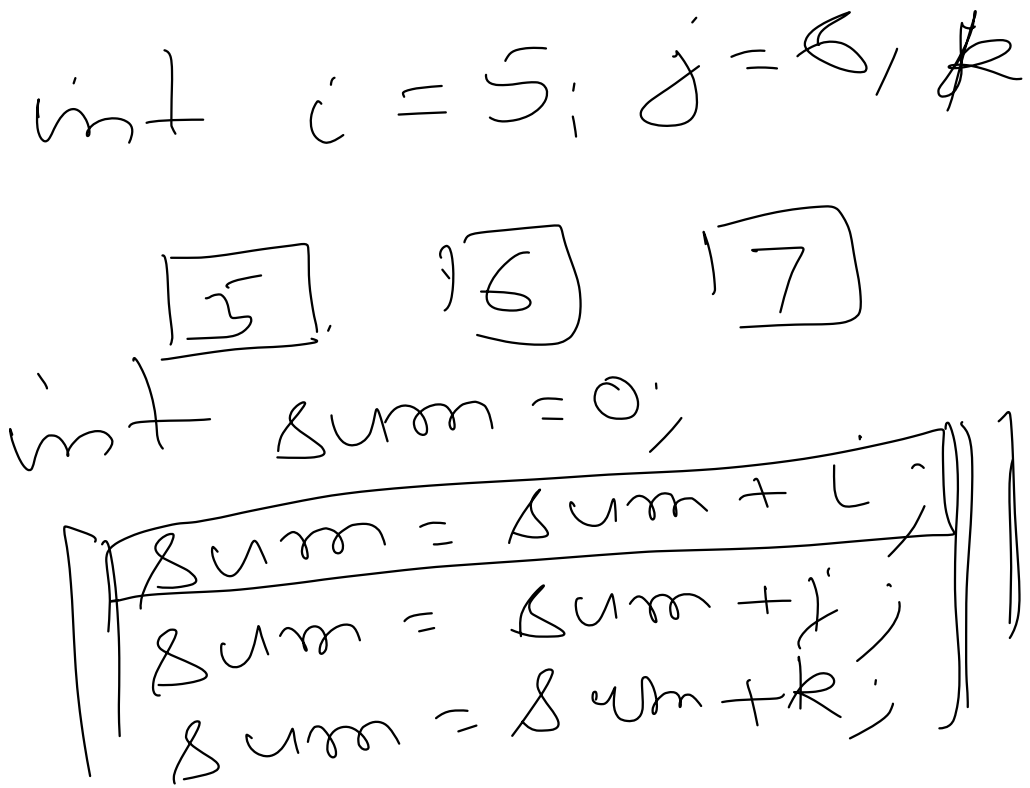
Data Structure

- A *data structure* is a way to **store** and **organize** data in order to facilitate **access** and **modifications**.

- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them

Array

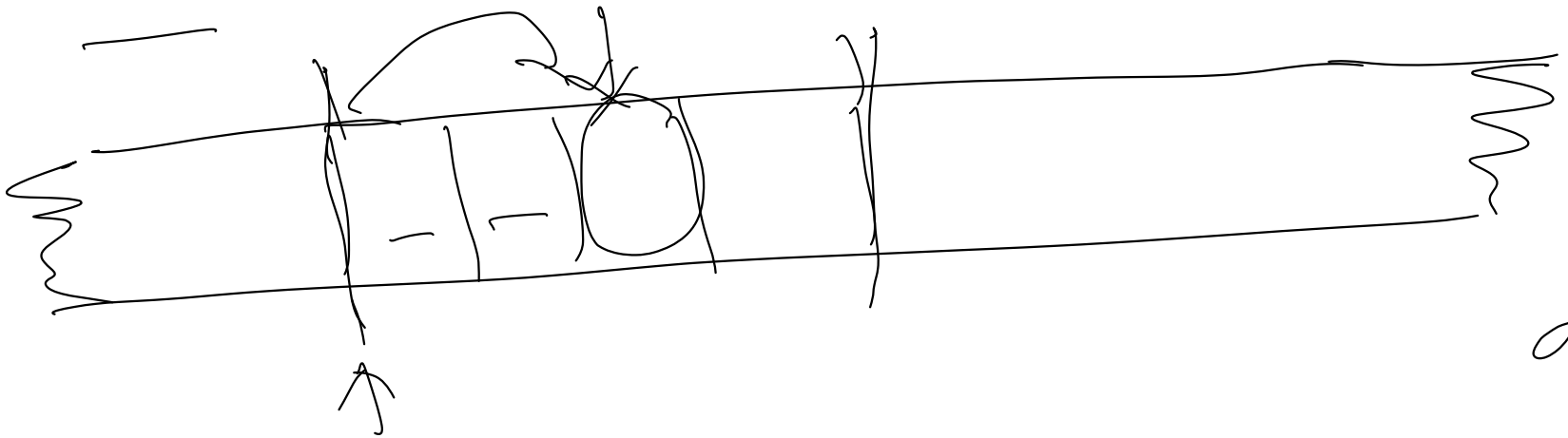
- Need for an array?



Properties of Array

char → 1 byte
int → 4 bytes

- Data Structure that stores multiple elements, all of the same type.
- All elements of an array are stored sequentially in memory, one after another.

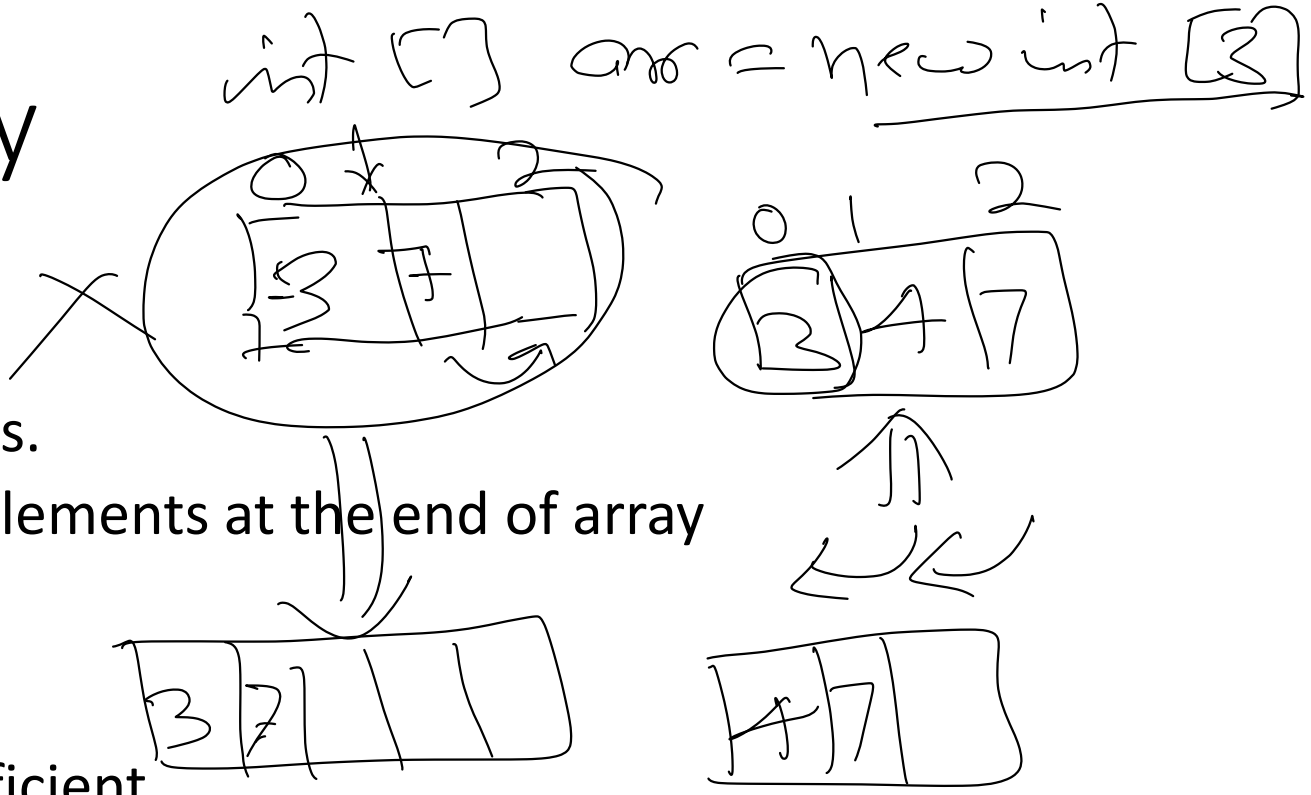


RAM

address

base address + i * memory of one box.

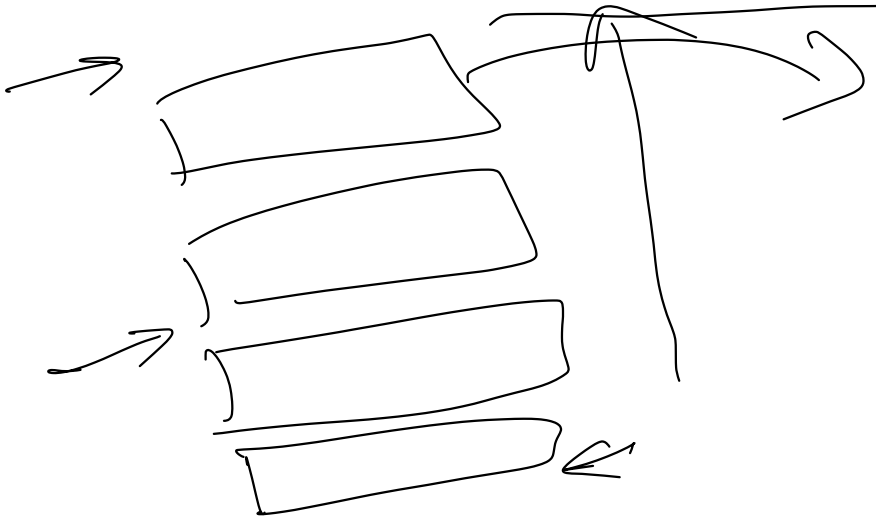
Pros and Cons of Array



- Advantages
  - Efficient lookup OR Random access.
  - Efficient in adding and removing elements at the end of array
- Disadvantages
  - Fixed size. Resizing of array is inefficient.
  - Insertion and deletion of elements, in middle of array is inefficient.

Stack

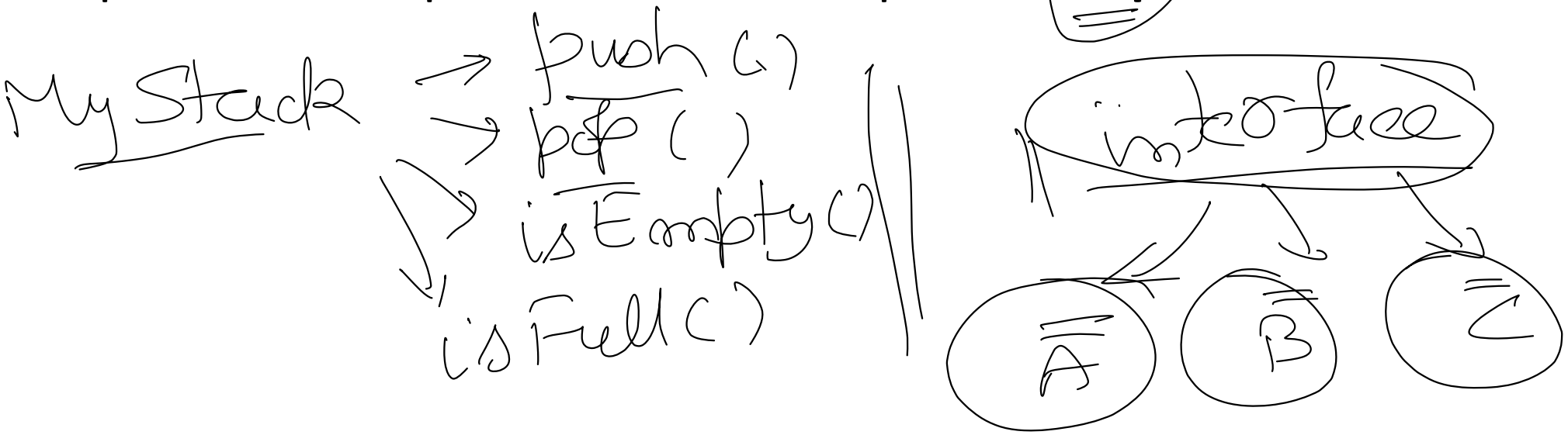
- Stack is a linear data structure.
- Stack is a container of objects.



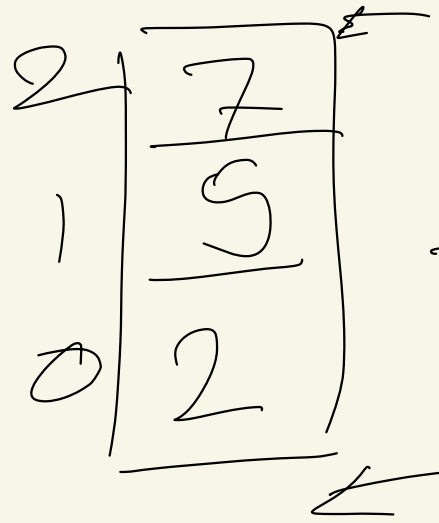
Stack operations

ADT
Abstract Data Type

- LIFO – Last In First Out
- Elements are added and removed according to LIFO principle.
- Operations are performed with respect to **"top"** of stack.



Size = 3



top = -1

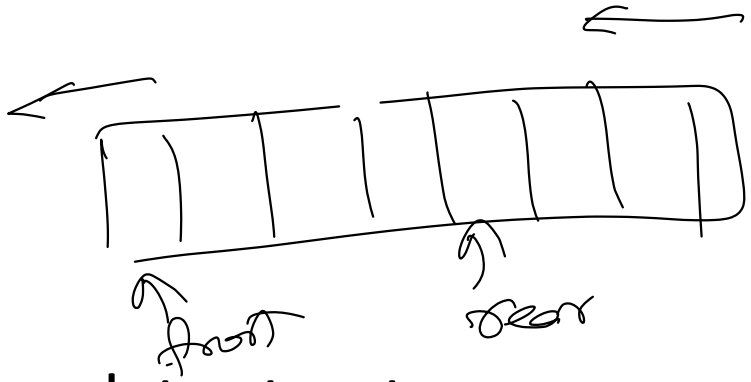
Push(2)
→ incr top
→ store elem at top

Push(5)
Push(7)
Push(10)

top

(size - 1) top

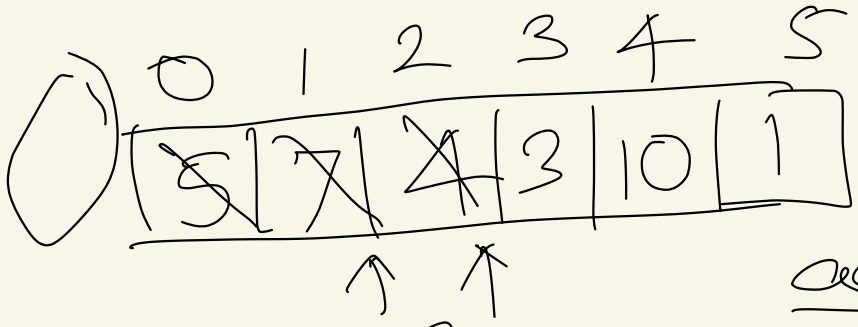
Queue



- Queue is a linear data structure.
- Queue is a container of objects.

Queue operations

- FIFO – First In First Out
- Elements are added and removed according to FIFO principle.
- Addition of elements are performed at “**rear**” of queue.
- Elements are removed from “**front**” of queue.



add

size = 6 → incr rear

front = ~~-1~~ 0 → add elem at rear

rear = ~~-1~~ 0 ~~1~~ 2 ~~3~~ 4 5

delete

isEmpty? → incr front

front == rear → remove elem from front

isFull

rear == size - 1

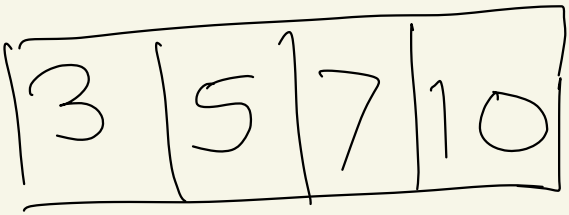
add: 5, 7, 4, 3, 10, 1

delete → 5, 7, 4

Size = 4 F →

f → ~~0~~ 1 ~~2~~ 3

r → ~~0~~ 1 ~~2~~ 3



→ R

add (3)

add (5)

delete() → 3

add (7)

add (10)

add (20) X Q FULL

delete() → 5

delete() → 7

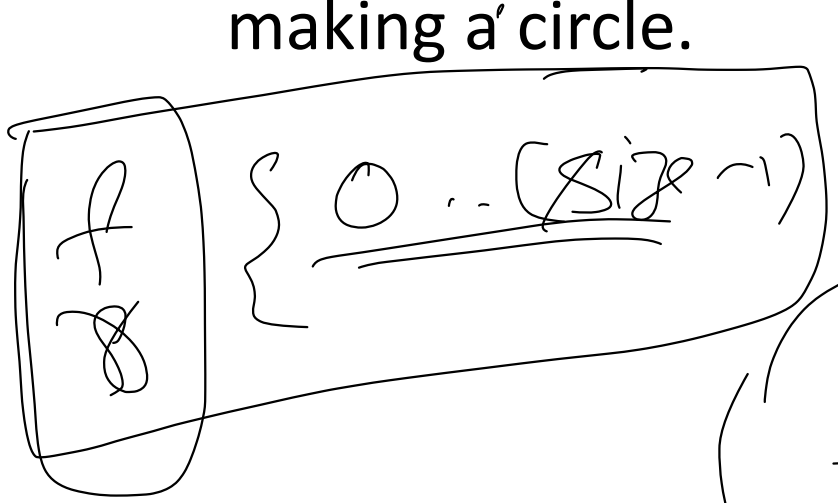
delete() → 10

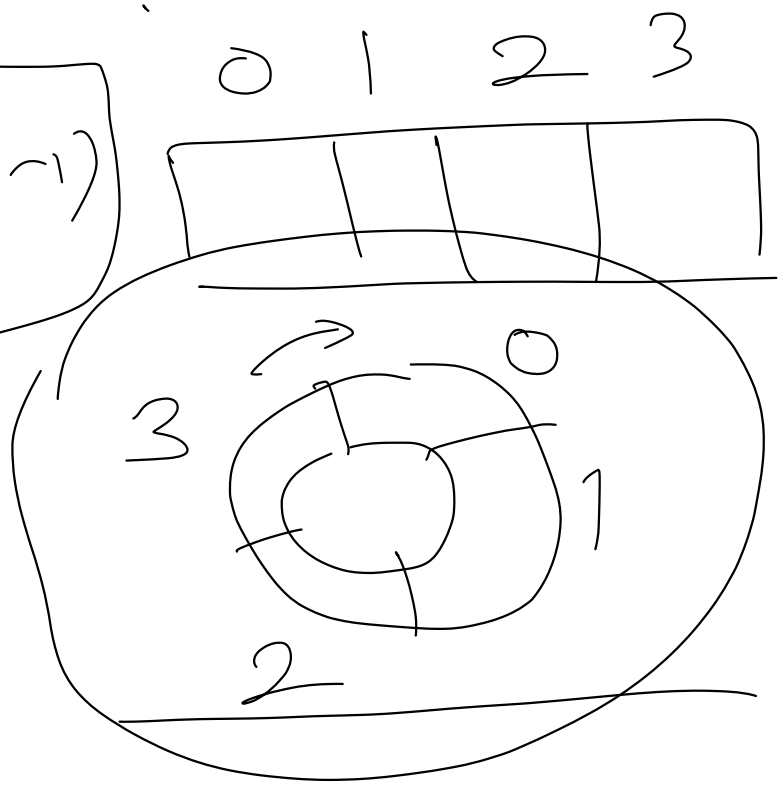
delete() X Q EMPTY

add (1) X Q FULL

Circular Queue

- Queue is a linear data structure.
- Last position of Circular Queue is connected back to first position, making a circle.





$x \bmod N \Rightarrow [0 \ldots N-1]$

$5 \bmod 2$

$2\overline{)5}\,(2$

4

1

$4\overline{)4}\,(1$

4

0

$rear = (rear + 1) \bmod size$

delete

→ fr = (f+1) mod size

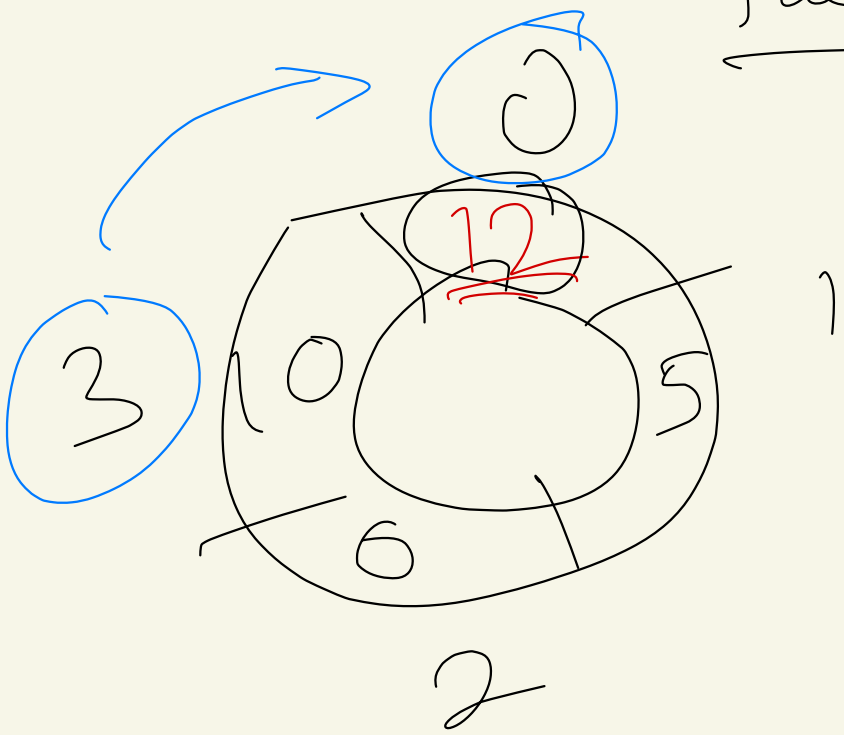
→ remove element from front

isEmpty

front == rear

isFull

(rear+1) mod size == front



f → 0

0 → 0 1 2 3 0

add → 5 6 10 12

interface QueueIntf

{

addQ

deleteQ

isEmpty

isFull

}

class A implements QueueIntf

class B implements QueueIntf

Circular Queue

- Queue is a linear data structure.
- Last position of Circular Queue is connected back to first position, making a circle.