

CSN 102: DATA STRUCTURES

Queue: Queue Fundamentals, Application of queue

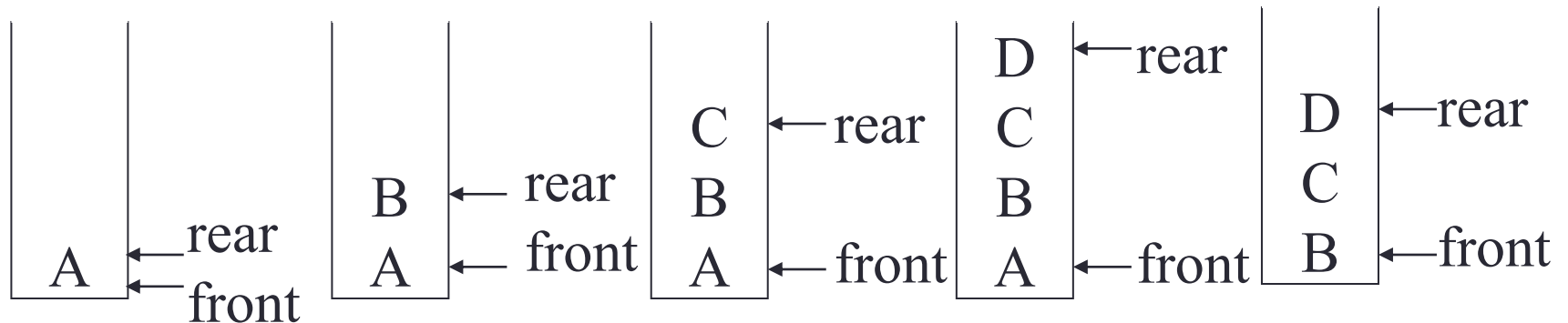
Abstract Data Type (ADT)

- An Abstract Data Type is:
 - Set of values
 - Set of operations which can be uniformly applied to these values
 - Set of Axioms

What is Queue?

- Stores the elements in particular way
- First In First Out (FIFO)
- Two pointers: Front and Rear
- enqueue(key): inserts the element key at Rear of queue
- dequeue(): deletes the element from front of the queue

First In First Out (FIFO)



Queue Abstract Data Type

- Queue ADT has:
 - Values based on what kind of data queue stores
 - Main operations:
 - `new()` : creates a new queue
 - `enqueue(Q, key)`: inserts element key at rear of queue Q
 - `dequeue(Q)`: deletes element from front of queue Q
 - `front(Q)`: returns the front element of queue Q without deleting it
 - Supported operations:
 - `isEmpty(Q)`: checks whether queue Q is empty or not
 - `isFull(Q)`: checks whether queue Q is full or not
 - `size(Q)`: returns the number of objects in queue Q

Queue Abstract Data Type

- Queue ADT has:
 - Axioms:
 - $\text{front}(\text{enqueue}(\text{new}(), v)) = v$
 - $\text{dequeue}(\text{enqueue}(\text{new}(), v)) = \text{new}()$
 - $\text{Front}(\text{enqueue}(\text{enqueue}(Q, w), v)) = \text{front}(\text{enqueue}(Q, w))$
 - $\text{Dequeue}(\text{enqueue}(\text{enqueue}(Q, w), v)) = \text{enqueue}(\text{dequeue}(\text{enqueue}(Q, w)), v)$

Queue Operations

enqueue(Q, key)

if (queue is not full)

increase rear by 1

insert key at rear

dequeue(Q)

if (queue is not empty)

key = delete element from front

increase front by 1

return (key)

Queue application: Job Scheduling

- Single processor and more than one job wants to execute
- More jobs are entering the system while other executing
- Once a job/process is executed, no longer required to be stored
- Eg. Printing documents using a printer
- Some strategy is required to execute all the processes

First Come First Serve (FCFS)

- The job which enters the system first, will be executed first
 - Once finished execution, execute next job in the queue
 - Eg. Print file1, then file2 and so on
 - Implemented using a Queue
-
- Start executing the first job in Queue
 - Insert new jobs to the end of Queue
 - Once execution is done, get the next job from front and start execution of this job

Job Scheduling: Example

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					Queue Q is empty
0	0	J1				Job J1 added to Q
0	1	J1	J2			Job J2 added to Q
1	1		J2			Job J1 deleted from Q
1	2		J2	J3		Job J3 added to Q
1	3		J2	J3	J4	Job J4 added to Q
2	3			J3	J4	Job J2 deleted from Q

Array implementation of Queue(1)

front \leftarrow -1;

rear \leftarrow -1;

isFull()

if (rear = N-1)

return true;

else

return false;

size()

if (front = -1)

return 0

else

return (rear + 1 – front)

isEmpty()

if (!size() or front = rear + 1)

return true;

else

return false;

Array Implementation of Queue(2)

enqueue(key)

if (isFull())

 “Queue is full”

else if (front = -1)

 front \leftarrow 0;

 rear \leftarrow 0;

 Q[rear] \leftarrow key;

else

 rear \leftarrow rear + 1;

 Q[rear] \leftarrow key;

Array Implementation of Queue(3)

dequeue()

if (isEmpty())

“Queue is empty”

else

key \leftarrow Q[front]

front \leftarrow front + 1;

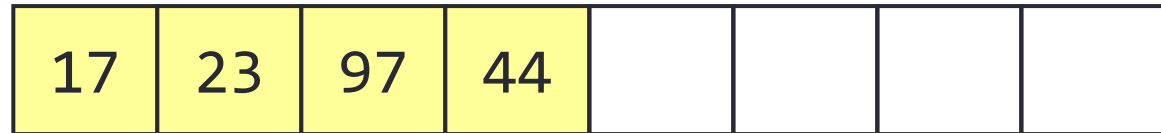
return key

Sample

front = 0

rear = 3

Initial queue:



After insertion:



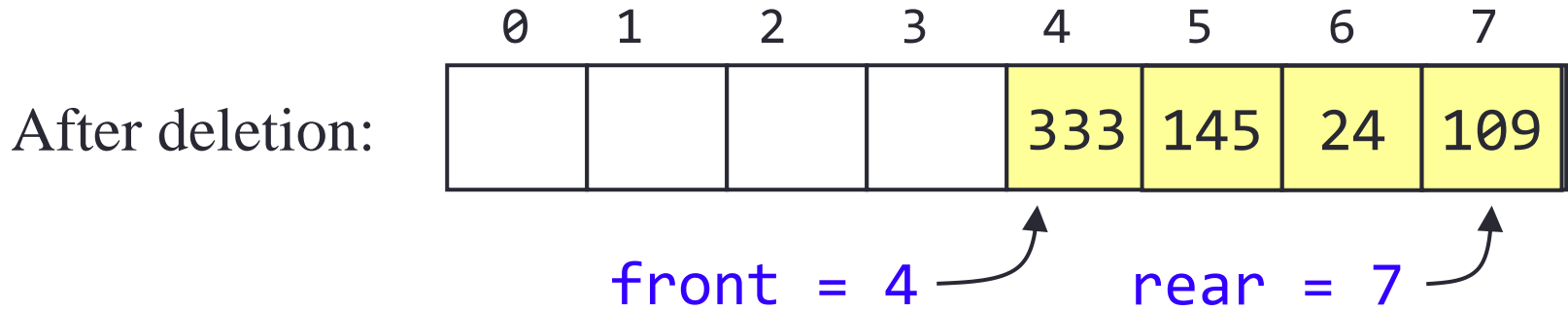
After deletion:

front = 1

rear = 4

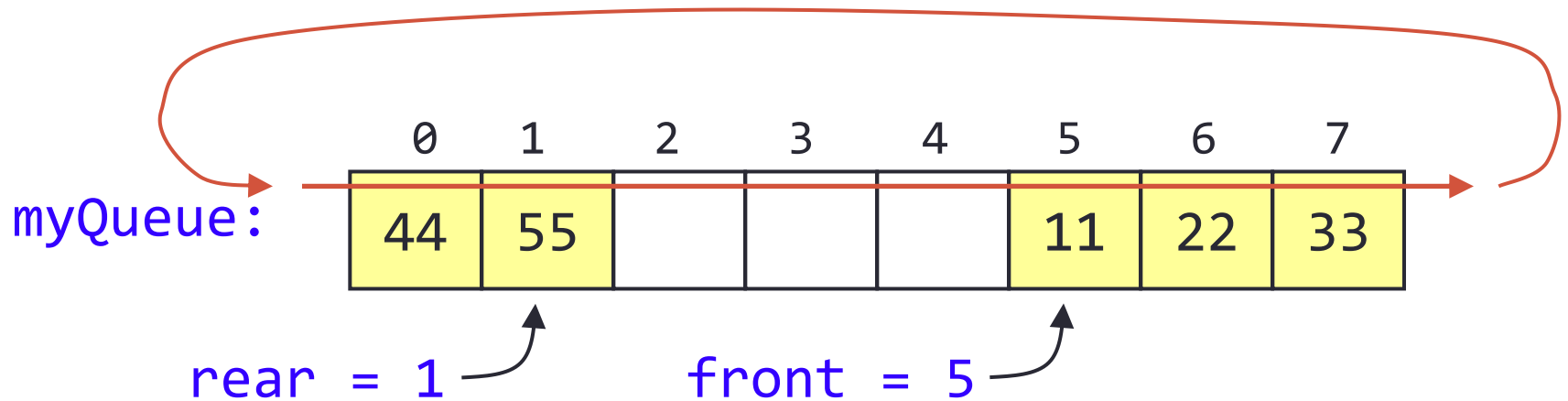


Issue with Implementation



- Problem: Even if space is available, can't insert the objects in queue
- Solution: circular queue

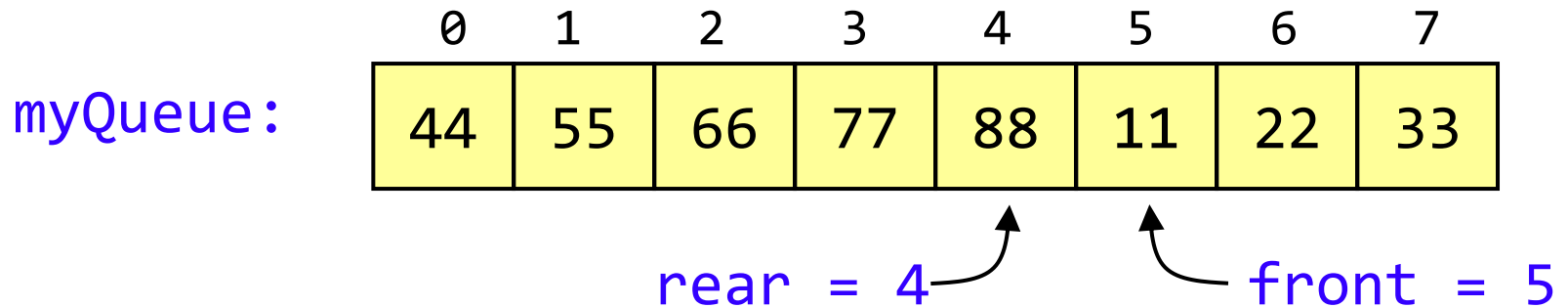
Circular Queue



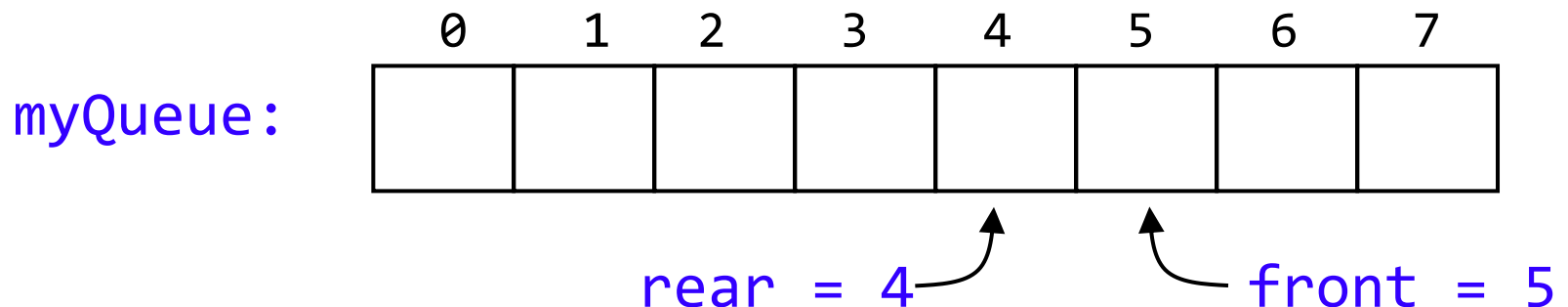
- Once end of array is reached, start inserting/deleting from the beginning of array
- Updated dequeue: $\text{front} = (\text{front} + 1) \% \text{length};$
- updated enqueue: $\text{rear} = (\text{rear} + 1) \% \text{length};$

Full and empty queues

- If the queue were to become completely full, it would look like this:

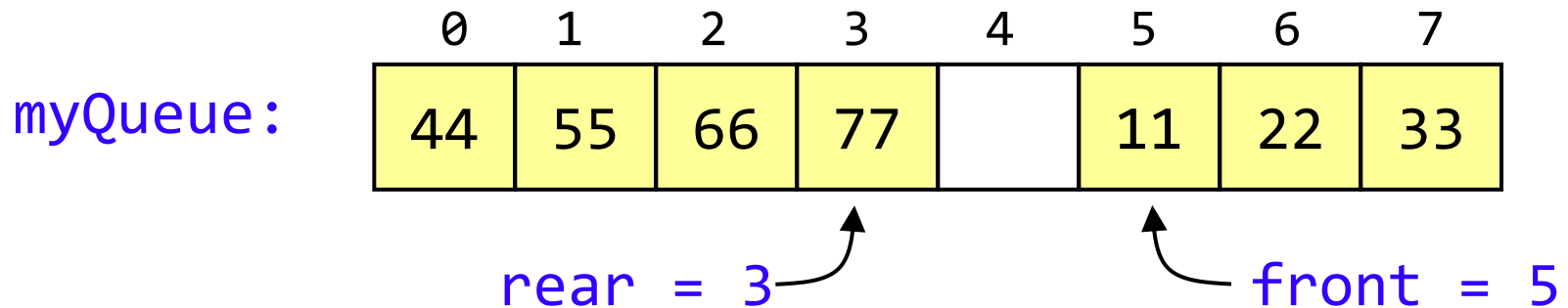


- If we were then to remove all eight elements, making the queue completely empty, it would look like this:



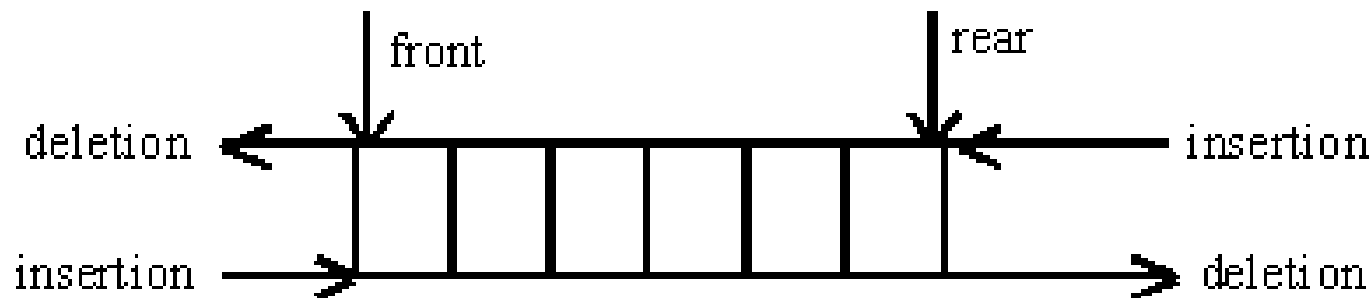
Issue with Implementation

- Problem: Both full and empty queue has same front and rear values
- Solution: consider queue full when it has $n-1$ elements



Double Ended Queue(Deque)

- Insertion and deletion can happen at both ends of the queue
- Separate function for insertion and deletion from front and rear



Implementation of Dequeue

```
insert_F(queue Q, int data) { //insert in front of queue
    if (Q is full)
        print ("overflow");
    else
        front = front-1;
        Q[front] = data; }
```

```
delete_F(queue Q) { //delete from front of queue
    if (Q is empty)
        print ("underflow");
    else
        temp = Q[front];
        front = front +1;
        return temp; }
```

Implementation of Dequeue

```
insert_R(queue Q, int data) { //insert in rear of queue
    if (Q is full)
        print ("overflow");
    else
        rear = rear + 1;
        Q[rear] = data; }
```

```
delete_R(queue Q) { //delete from rear of queue
    if (Q is empty)
        print ("underflow");
    else
        temp = Q[rear];
        rear = rear - 1;
        return temp; }
```

Versions of Deque

- **Input restricted Deque:** deletion can be made from both ends, but insertion can be made at one end only.
- Application: Web Browser History
- **Output restricted Deque:** insertion can be made at both ends, but deletion can be made from one end only.