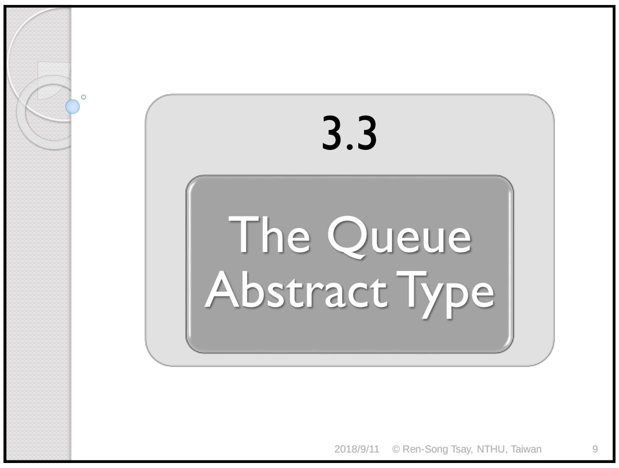
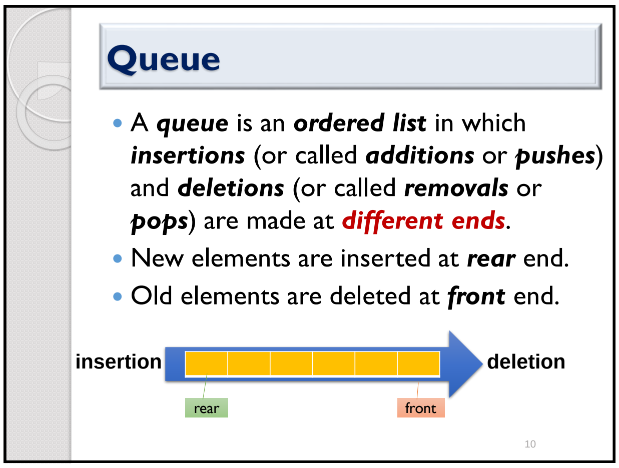


3.3

The Queue Abstract Type

2018/9/11 © Ren-Song Tsay, NTHU, Taiwan 9

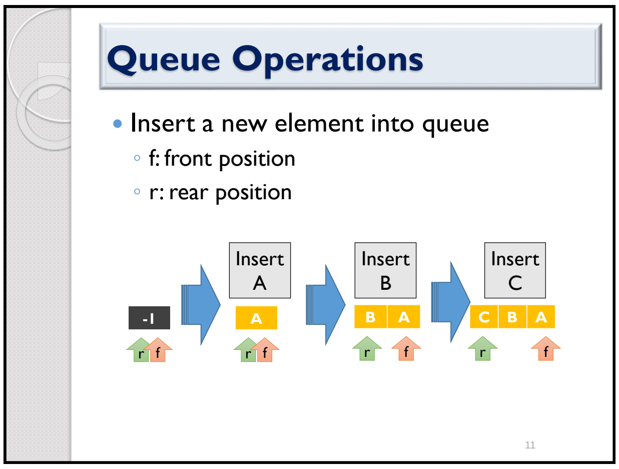


Queue

- A ***queue*** is an ***ordered list*** in which ***insertions*** (or called ***additions*** or ***pushes***) and ***deletions*** (or called ***removals*** or ***pops***) are made at ***different ends***.
- New elements are inserted at ***rear*** end.
- Old elements are deleted at ***front*** end.

insertion
deletion
rear
front

10

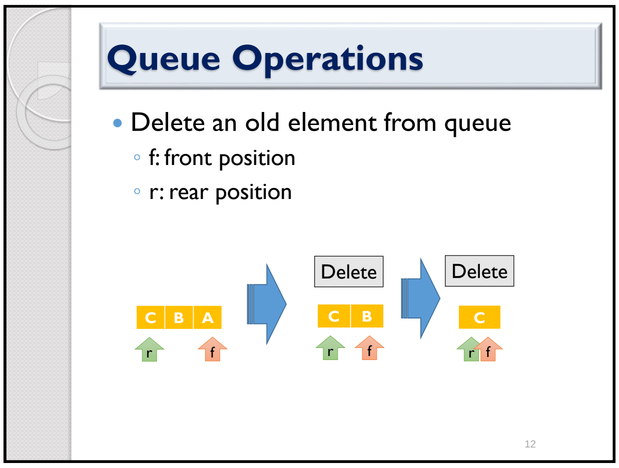


Queue Operations

- Insert a new element into queue
  - f: front position
  - r: rear position

-1
r f
Insert A
A
r f
Insert B
B A
r f
Insert C
C B A
r f

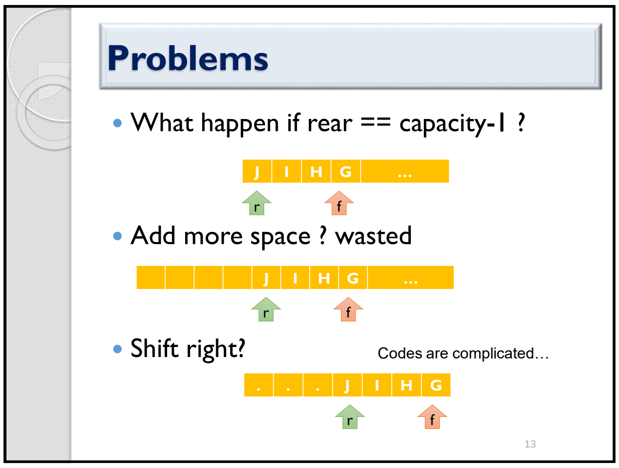
11



Queue Operations

- Delete an old element from queue
  - f: front position
  - r: rear position

12

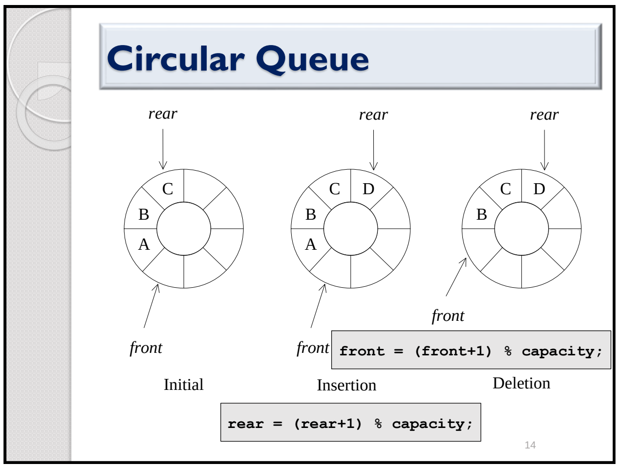


Problems

- What happen if rear == capacity-1 ?
- Add more space ? wasted
- Shift right?

Codes are complicated…

13

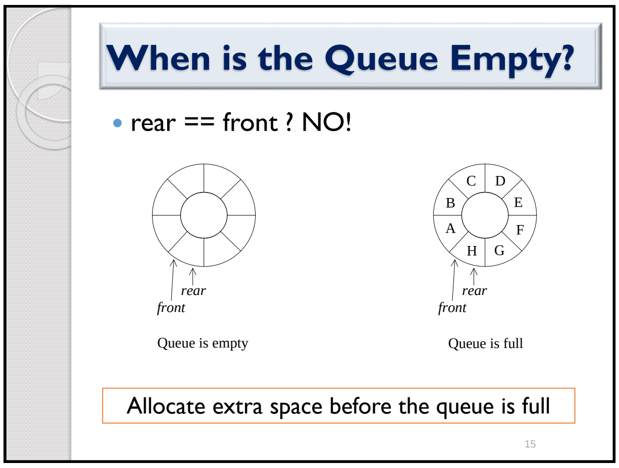


Circular Queue

```
front = (front+1) % capacity;
```

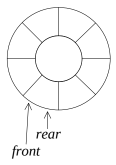
```
rear = (rear+1) % capacity;
```

14

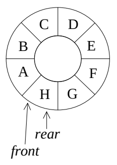


When is the Queue Empty?

- rear == front ? NO!



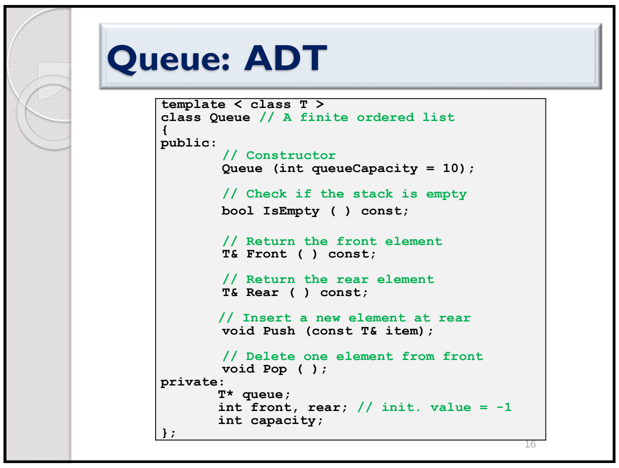
Queue is empty



Queue is full

Allocate extra space before the queue is full

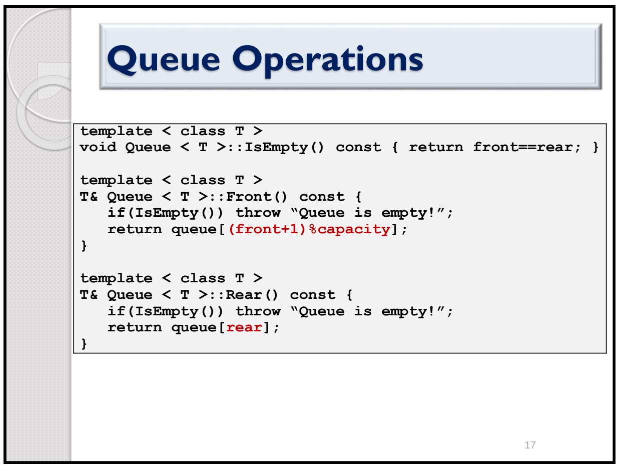
15



Queue: ADT

```
template < class T >
class Queue // A finite ordered list
{
public:
        // Constructor
        Queue (int queueCapacity = 10);

        // Check if the stack is empty
        bool IsEmpty ( ) const;

        // Return the front element
        T& Front ( ) const;

        // Return the rear element
        T& Rear ( ) const;

       // Insert a new element at rear
        void Push (const T& item);

        // Delete one element from front
        void Pop ( );
private:
       T* queue;
       int front, rear; // init. value = -1
       int capacity;
};
```

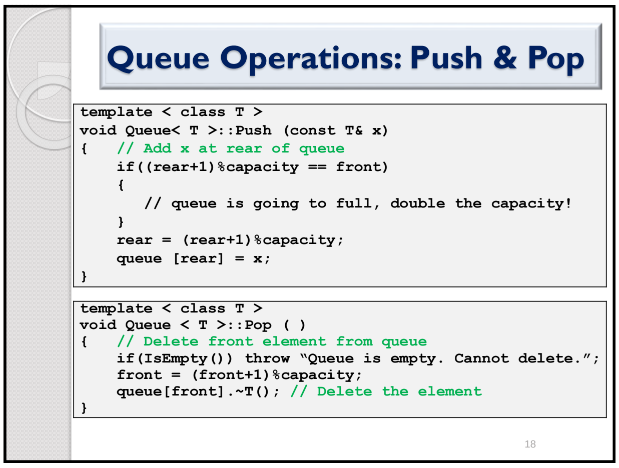
16



Queue Operations

```
template < class T >
void Queue < T >::IsEmpty() const { return front==rear; }

template < class T >
T& Queue < T >::Front() const {
   if(IsEmpty()) throw "Queue is empty!";
   return queue[(front+1)%capacity];
}

template < class T >
T& Queue < T >::Rear() const {
   if(IsEmpty()) throw "Queue is empty!";
   return queue[rear];
}
```

17

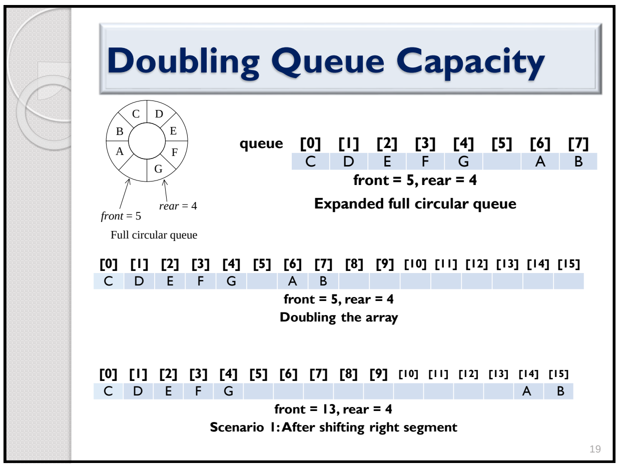


Queue Operations: Push & Pop

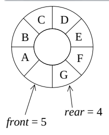
```
template < class T >
void Queue< T >::Push (const T& x)
{   // Add x at rear of queue
    if((rear+1)%capacity == front)
    {
       // queue is going to full, double the capacity!
    }
    rear = (rear+1)%capacity;
    queue [rear] = x;
}
```

```
template < class T >
void Queue < T >::Pop ( )
{   // Delete front element from queue
    if(IsEmpty()) throw "Queue is empty. Cannot delete.";
    front = (front+1)%capacity;
    queue[front].~T(); // Delete the element
}
```

18



Doubling Queue Capacity



Full circular queue

queue	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	C	D	E	F	G		A	B

front = 5, rear = 4

Expanded full circular queue

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
C	D	E	F	G		A	B								

front = 5, rear = 4

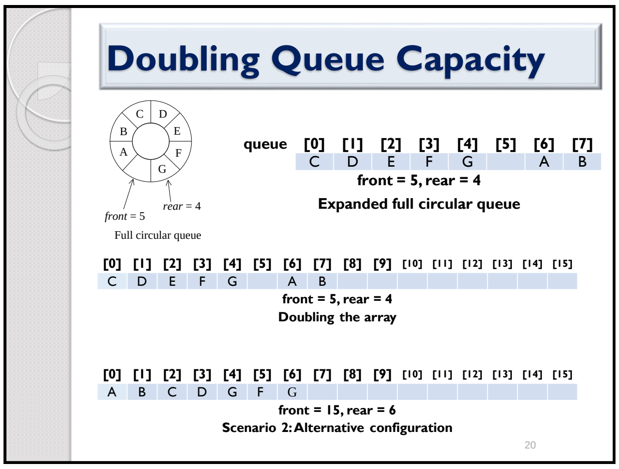
Doubling the array

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
C	D	E	F	G										A	B

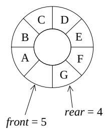
front = 13, rear = 4

Scenario 1: After shifting right segment

19



Doubling Queue Capacity



Full circular queue

queue	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	C	D	E	F	G		A	B

front = 5, rear = 4

Expanded full circular queue

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
C	D	E	F	G		A	B								

front = 5, rear = 4

Doubling the array

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
A	B	C	D	G	F	G									

front = 15, rear = 6

Scenario 2: Alternative configuration

20