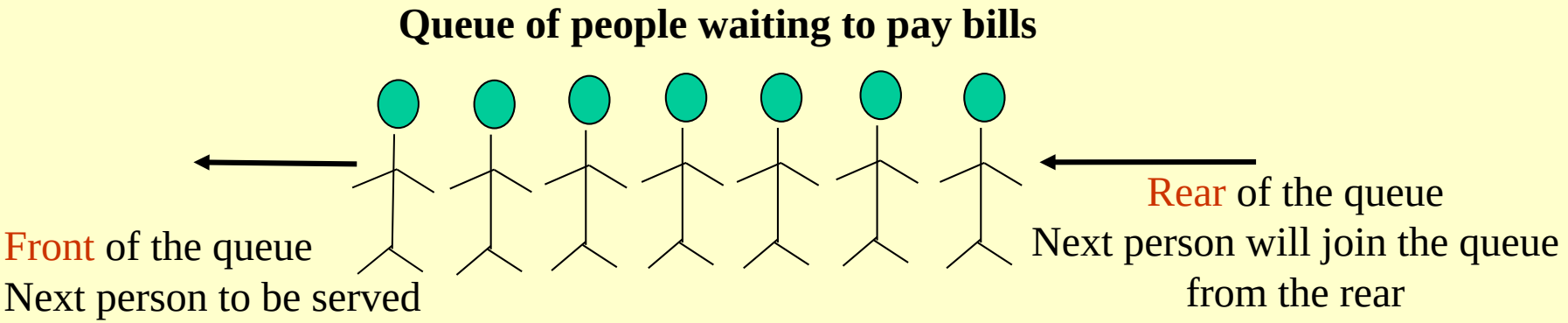
Queues – Chapter 3

- A queue is a data structure in which all additions are made at one end called the rear of the queue and all deletions are made from the other end called the front of the queue

- Alternatively, in a queue the element deleted is the one that stayed in the queue the longest. This is also called first-in-first-out (FIFO)

- Classical example of queues is a queue of people waiting to pay bills.

Queue Concept and Example



- A queue has a Front and a Rear
- The insert operation is often called Enqueue
- The delete operation is often called Dequeue

Queue ADT

- A queue is a data structure in which all additions are made at one end called the rear of the queue and all deletions are made from the other end called the front of the queue

- Common queue operations:
  - Enqueue(item) – Add the item to the end of the Q
  - Dequeue() – Remove & return the item at the front
  - isEmpty() – Return true if the Q is empty
  - isFull() – Return true if the Q is full

How do we implement queue ADT?

- 2 ways to implement a queue
  - Using an array
  - Using a linked list

Array Implementation of Queues: Operations

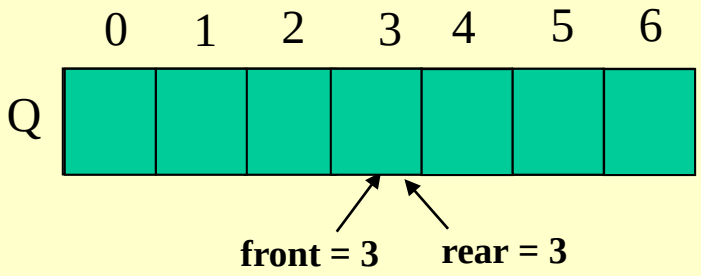
- Use an array int Q[N]
- front points to the front element of the queue, i.e., holds the index of the array Q that contains the front of the queue
- rear points to next slot after the last element in the queue
- noItems contains the current number of items in the queue.
- An empty queue is one where noItems = 0
- A full queue is one where noItems = N

Array Implementation of Queues

- We can implement a queue of at most "N" elements with an array int Q[N] and 3 variables: int front, int rear, int noItems as follows

---

An Empty Queue

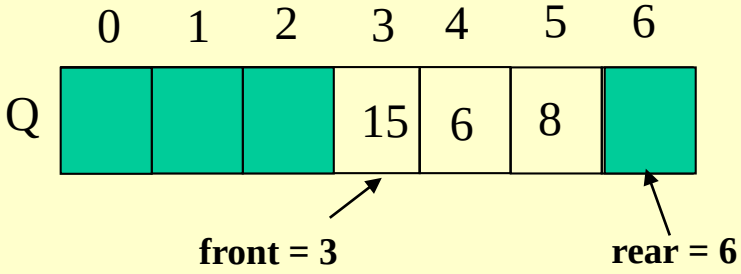


noItems = 0

- Front and rear are equal to each other and noItems = 0 in an empty Queue

---

A partially-full Queue

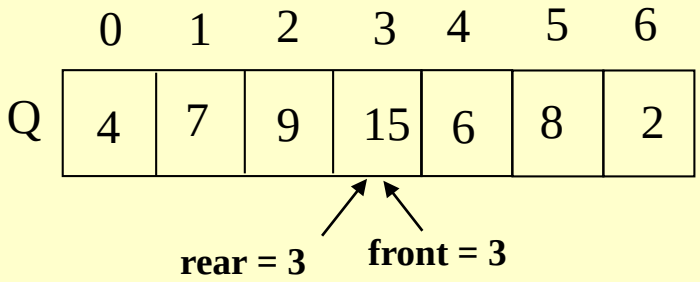


noItems = 3

- Front points to the first element in the Queue
- Rear points to the next slot after the last element in the Queue

---

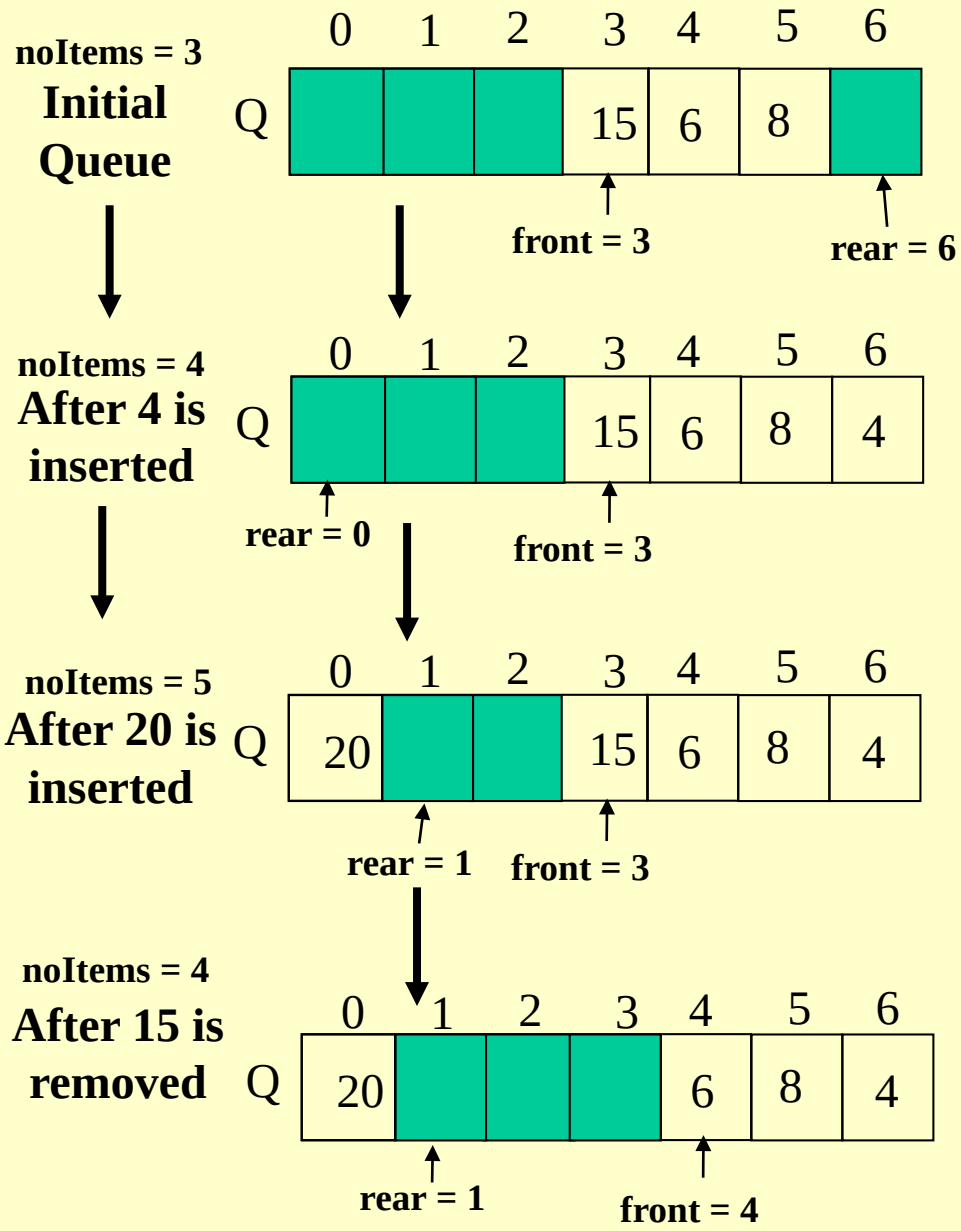
A Full Queue



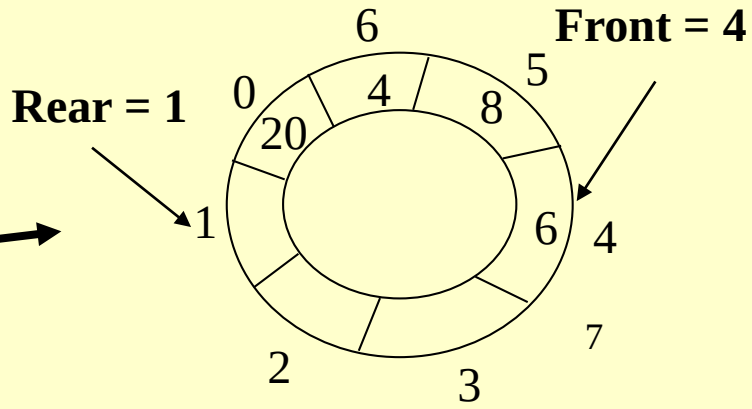
noItems = 7

- Front and rear are equal to each other and noItems = N=7 in a full Queue

Array Implementation of Queues



Conceptual view of the final queue: circular array



Queue Using Arrays: Declarations & Operations

```
class Queue {
private:
  static int N = 100; // size of the queue
  int Q[];     // Array holding the queue elements (ints)
  int front;   // front of the queue
  int rear;    // rear of the queue
  int noItems; // # of items in the queue

public:
  Queue();
  bool isEmpty();
  bool isFull();
  int Enqueue(int item);
  int Dequeue();
};
```

Queue Operations: Constructor, isEmpty, isFull

```
// Constructor
Queue(){
  Q = new int[N];
  front = rear = noOfItems = 0;
} //end-Queue

// Returns true if the Q is empty
bool isEmpty(){
  return noOfItems == 0;
} //end-isEmpty

// Returns true if the Q is full
bool isFull(){
  return noOfItems == N;
} //end-isFull
```

Queue Operations: Enqueue

```
// Inserts a new item into the queue
// Returns 0 on success, -1 on failure
int Enqueue(int newItem){
  if (isFull()){
    println("Queue is full");
    return -1;
  } //end-if

  Q[rear] = newItem;   // Put the new item at the end
  rear++; if (rear == N) rear = 0;  // Now move rear
  noItems++;  // One more item in the queue

  return 0;
} //end-EnQueue
```

Queue Operations: Dequeue

```
// Removes and returns the item at the front of the queue
// If the queue is empty, returns -1 (an error)
int Dequeue(){
  int idx = -1;

  if (isEmpty()){
    println("Queue is empty");
    return -1;
  } //end-if

  idx = front;   // This is where the first item is.
  front++; if (front == N) front = 0;   // Move front
  noItems--;   // One less item in the Queue

  return Q[idx];  // Return the item
} //end-Dequeue
```

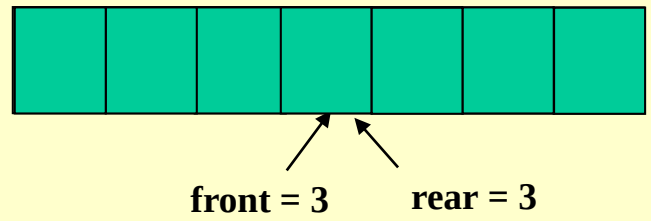
Queue Usage Example

```
main(){
  Queue q = new Queue();

  if (q.isEmpty()) println("Queue is empty");   // Q empty now

  q.Enqueue(49);
  q.Enqueue(23);

  println("Front of the Q is: " + q.Dequeue()); // prints 49
  q.Enqueue(44);
  q.Enqueue(22);

  println("Front of the Q is: " + q.Dequeue()); // prints 23
  println("Front of the Q is: " + q.Dequeue()); // prints 44
  println("Front of the Q is: " + q.Dequeue()); // prints 22.
  println("Front of the Q is: " + q.Dequeue()); // prints -1

  if (q.isEmpty()) println("Queue is empty");   // Q empty now

} //end-main
```

Array Implementation of Queues: Last Word

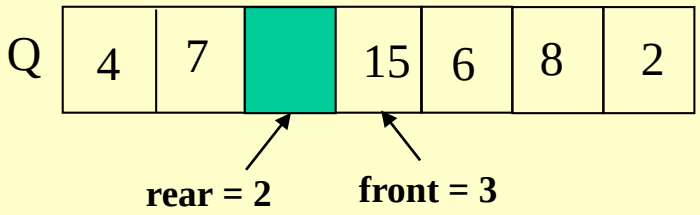
- We can implement a queue of at most "N-1" elements with an array int Q[N] and 2 variables: int front, int rear
- Think about how you would define
  - An empty queue
  - A full queue

An Empty Queue



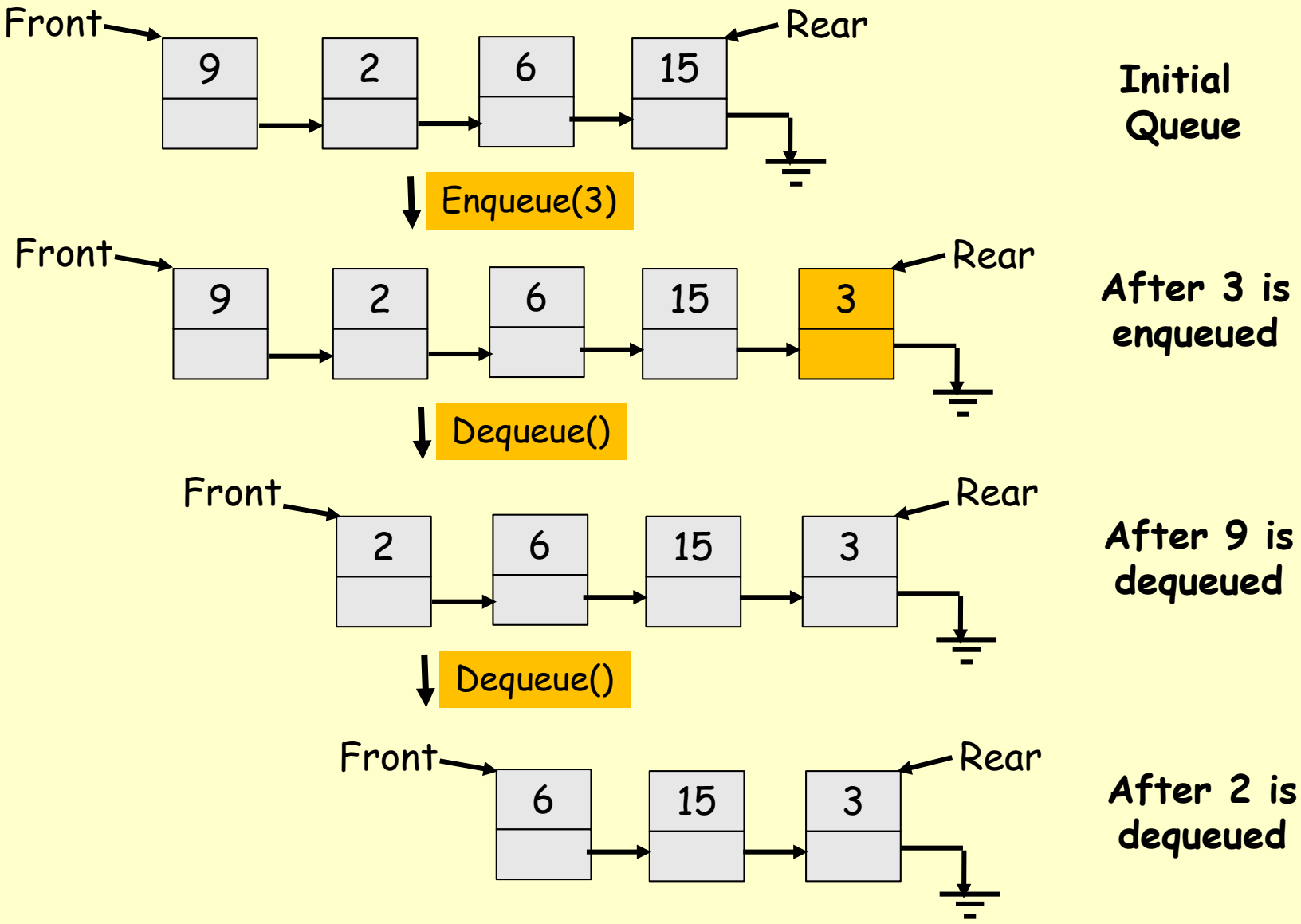
- Front and rear are equal to each other in an empty queue

A Full Queue



- There is one empty space between rear and front in a full queue

Linked-List implementation of Queues



Queue using Linked List: Declarations & Operations

```
class QueueNode {
  int item;
  QueueNode next;
  QueueNode(int e){item=e; next=null;}
}

class Queue{
private:
  QueueNode front; // Ptr to the front of the Q
  QueueNode rear;  // Ptr to the rear of the Q

public:
  Queue();
  bool isEmpty();
  void Enqueue(int item);
  int Dequeue();
};
```

Queue Operations: Constructor, isEmpty

```
// Constructor
Queue(){
  front = rear = null;
} //end-Queue

// Returns true if the Q is empty
bool isEmpty(){
  return front == null;
} //end-isEmpty
```

Queue Operations: Enqueue

```
// Inserts a new item into the queue
void Enqueue(int newItem){
  // Allocate a QueueNode for the item
  QueueNode node = new QueueNode(newItem);

  if (front == NULL){
    front = rear = node;
  } else {
    rear.next = node;
    rear = node;
  } //end-else
} //end-EnQueue
```

Queue Operations: Dequeue

```
// Removes and returns the item at the front of the queue
// If the queue is empty, returns -1 (an error)
int Dequeue(){
  if (isEmpty()){
    println("Queue is empty");
    return -1;
  } //end-if

  QueueNode tmp = front; // Keep a ptr to the front node
  front = front.next;    // Remove the front node
  if (front == null) rear = NULL; // Empty Q?

  // Return the item
  return tmp.item;
} //end-Dequeue
```

Queue Usage Example

```
main(){
  Queue q = new Queue();

  if (q.isEmpty()) println("Queue is empty");   // Q empty now

  q.Enqueue(49);
  q.Enqueue(23);

  println("Front of the Q is: " + q.Dequeue()); // prints 49
  q.Enqueue(44);
  q.Enqueue(22);

  println("Front of the Q is: " + q.Dequeue()); // prints 23
  println("Front of the Q is: " + q.Dequeue()); // prints 44
  println("Front of the Q is: " + q.Dequeue()); // prints 22.
  println("Front of the Q is: " + q.Dequeue()); // prints -1

  if (q.isEmpty()) println("Queue is empty");   // Q empty now

} //end-main
```

Applications of Queues

- **File servers**: Users needing access to their files on a shared file server machine are given access on a FIFO basis

- **Printer Queue**: Jobs submitted to a printer are printed in order of arrival

- Phone calls made to **customer service hotlines** are usually placed in a queue

- Expected wait-time of real-life queues such as customers on phone lines may be too hard to solve analytically use queue for **simulating real-life queues**