CS1102: Data Structures and Algorithms

Part 5

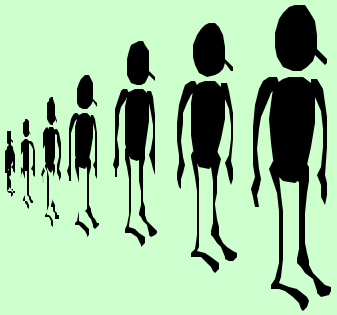
Queue

Zoltan KATO

S16 06-12

Adopted from Chin Wei Ngan's cs1102 lecture notes

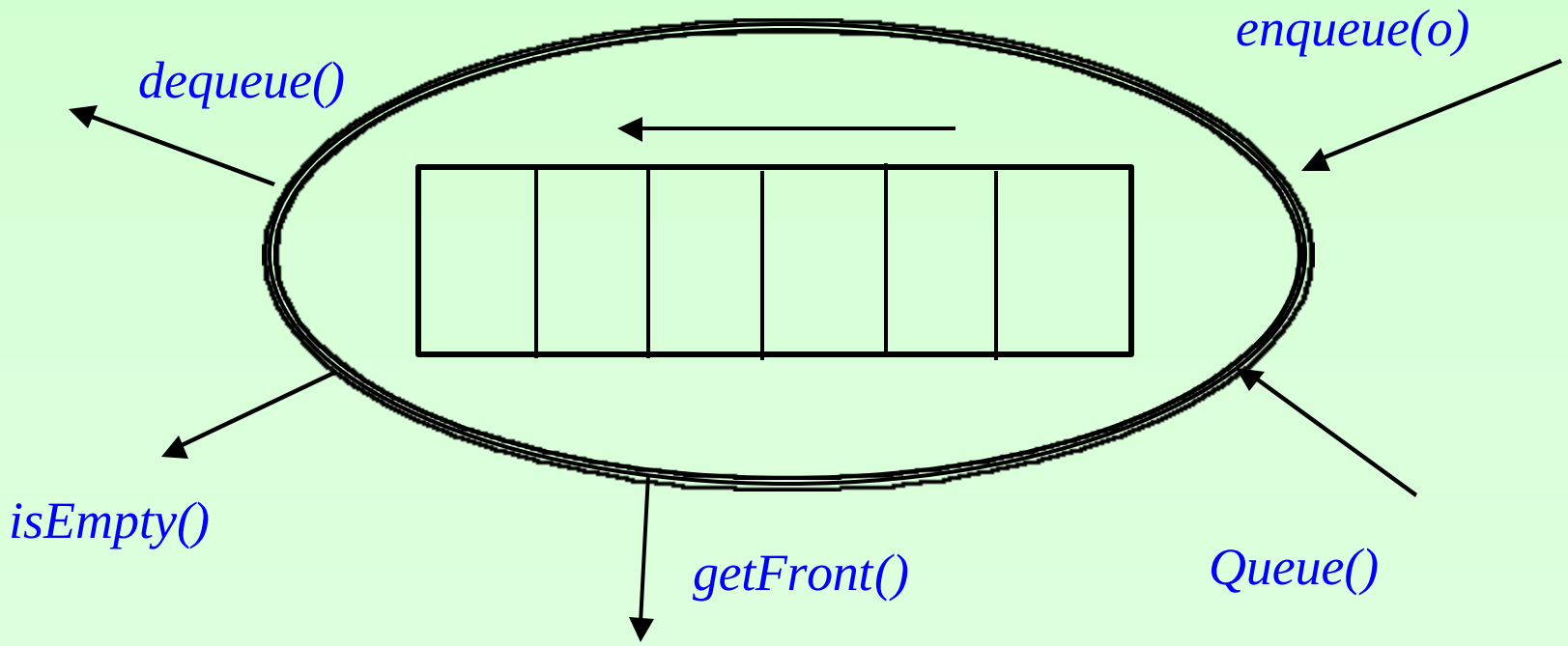
Queues



What is a Queue?

Queues implement the FIFO (first-in first-out) policy.

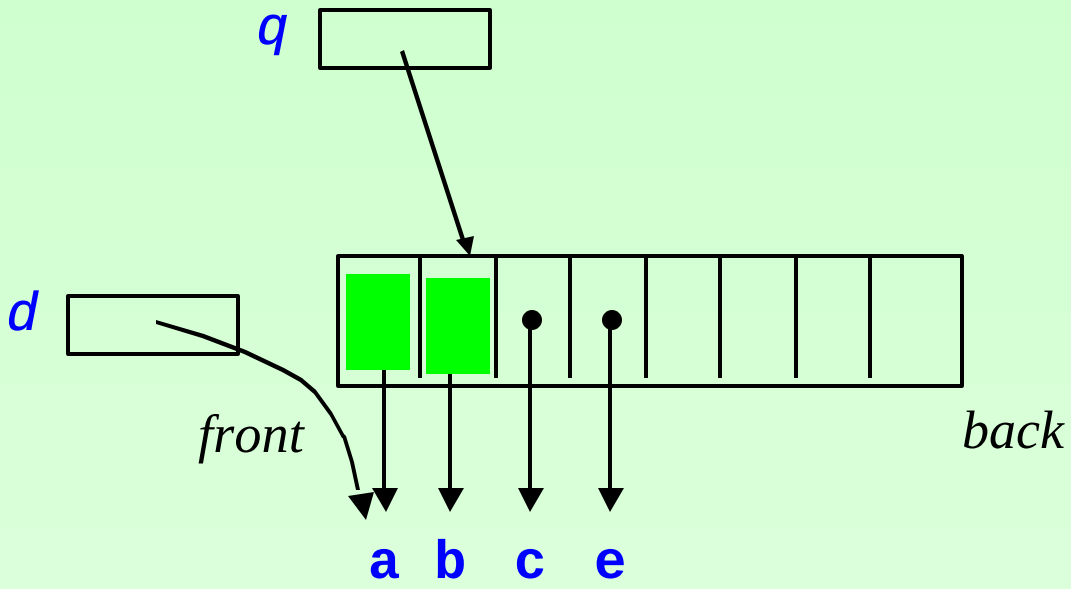
An example is the printer/job queue!



Sample Code

```
Queue q = makeQueue();
q.enqueue("a");
q.enqueue("b");
q.enqueue("c");
d=q.getFront();
q.dequeue();
q.enqueue("e");
q.dequeue();
```

What is a Queue?



Applications

Many application areas for Queues:

- *print queue*
- *simulation*
- *breath-first traversal of trees*
- *checking palindrome*

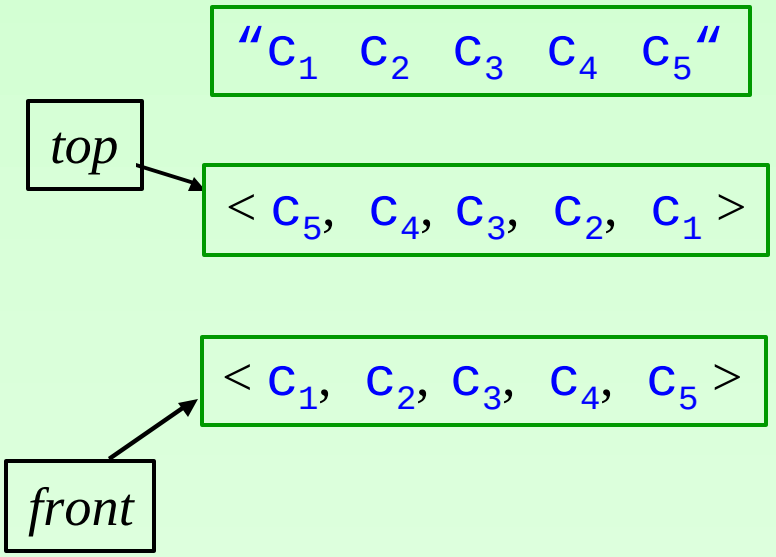
Recognising Palindrome

A string which reads the same either *left to right*, or *right to left* is known as a palindrome.

Palindromes : "r a d a r" and "d e e d".
Counter Example : "d a t a"

Procedure

- Given a string
- Stack to *reverse* the character order of string.
- Queue to *preserve* the character order of string.
- Check if the two sequences are the same.



Procedure Recognising Palindrome

```
public static boolean palindrome(String v) throws Exception{
 Stack s = makeStack();
 Queue q = makeQueue();
 // push string into stack, and also queue
 for (j=0; j<v.length(); j++)
  { Character c = Character(v.charAt(j));
    s.push(c);
    q.enqueue(c);
  }
 // push string into stack, and also queue
 boolean OKflag=true;
 while (!s.isEmpty() && OKflag)
  { Object vs=s.top();
    Object vq=q.getFront();
    if !(vs.equals(vq)) {OKflag = false;}
    s.pop(); q.dequeue();
  }
return OKflag;
}
```

Queue ADT

```
class Queue {
 ...
 public Queue() {...}
        // pre : true
        // post : this = < >
 public void enqueue(Object o) {...}
        // pre : this = <a1,...,an>
        // post : this = <a1,...,an,o>
 public void dequeue() throws Underflow {...}
        // pre : this = <a1,...,an>
        // post : this = <a2,...,an>
        //      : Underflow raised if n==0
 public Object getFront() throws Underflow {...}
        // pre : this = <a1,...,an>
        // post : returns a1
        //      : Underflow raised if n==0
 public boolean isEmpty() {...}
        // pre : this = <a1,...,an>
       // post : returns (n==0)
}
```

Java Interface — Queue ADT

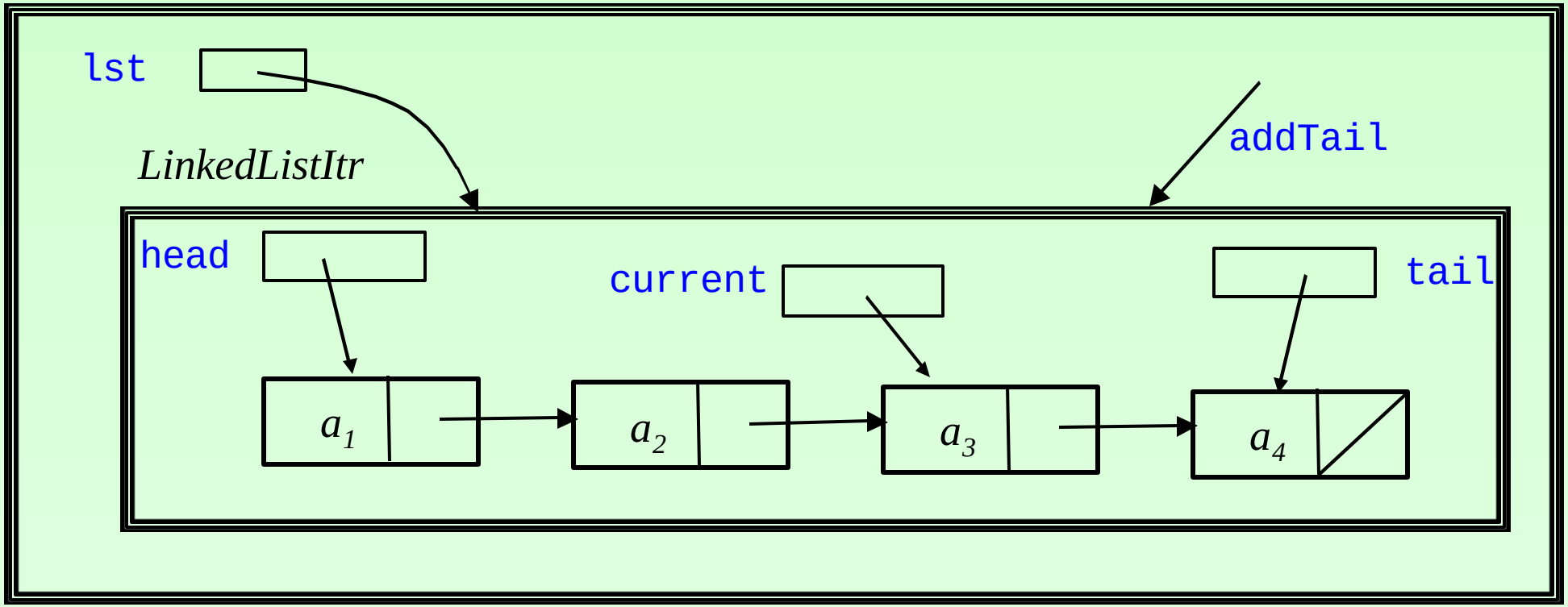
We can also use Java Interface to specify Queue ADT Interface. This provides a more abstract mechanism to support simultaneous implementations.

```
interface Queue {

 void enqueue(Object o)                // insert o to back of Q

 void dequeue() throws Underflow;      // remove oldest item

 Object getFront() throws Underflow;   // retrieve oldest item

 boolean isEmpty();                    // checks if Q is empty

}
```

Implementation of Queue (Linked-List)

Can use `LinkedListItr` as underlying implementation of Queues

Queue



Codes Implementation of Queue (Linked-List)

```
class QueueLL implements Queue {

 private LinkedListItr lst;

 public QueueLL() { lst = new LinkedListItr(); }

 public static Queue makeQueue()              // return a new empty queue
  { return new QueueLL();  }


 public void enqueue(Object o)                // add o to back of queue
  { lst.addTail(o); }

 public void dequeue() throws Underflow      // remove oldest item
  { try {lst.deleteHead();
    catch (ItemNotFound e)
        {throw new Underflow("dequeue fails - empty q";};
  }
```

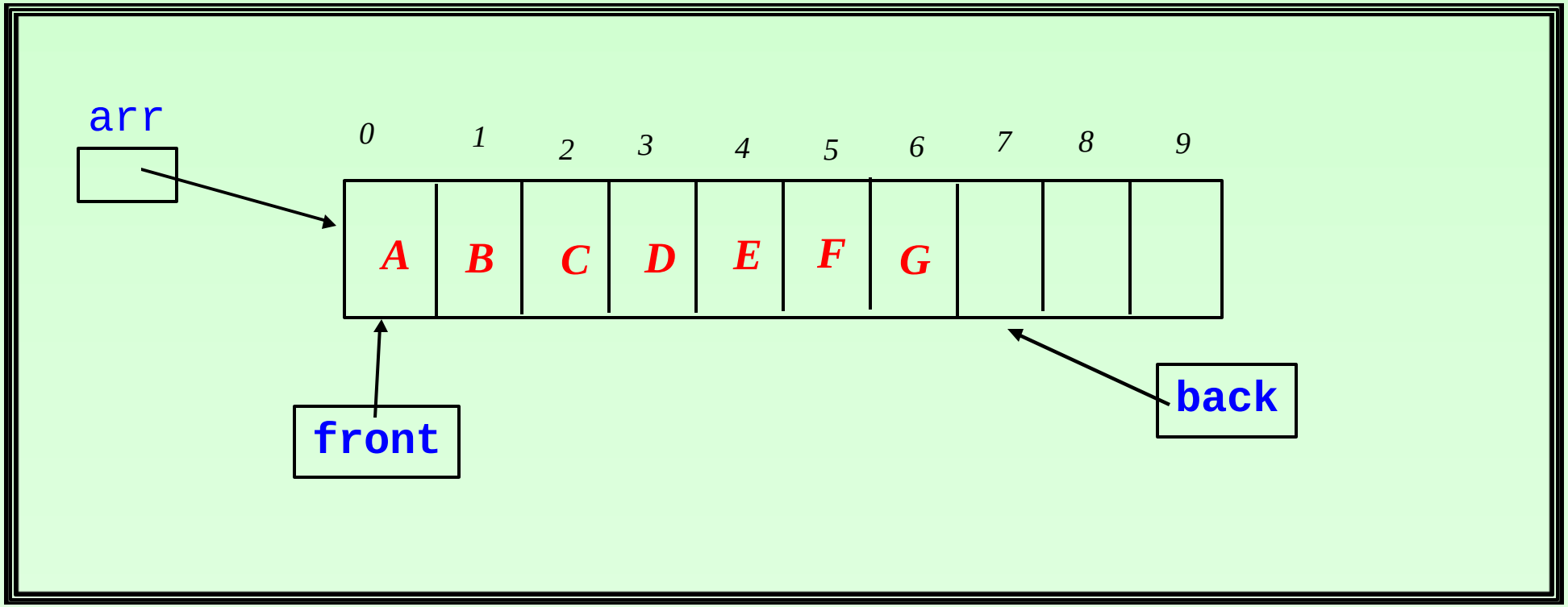
Codes Implementation of Queue (Linked-List)

```
public Object getFront() throws Underflow;   // retrieve oldest item
  { try { lst.first();
          return lst.retrieve();
    } catch (ItemNotFound e)
        {throw new Underflow("getFront fails - empty queue");};
  }

public boolean isEmpty()                     // return true if empty
  { return lst.isEmpty();  }
```

Implementation of Queue (Array)

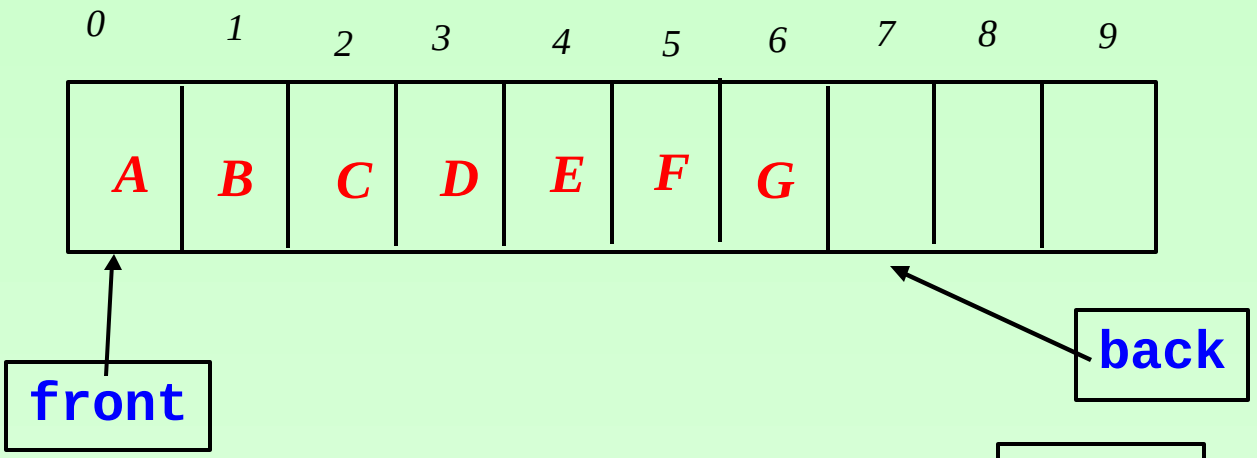
Can use `Array` with `front` and `back` pointers as implementation of queue

Queue

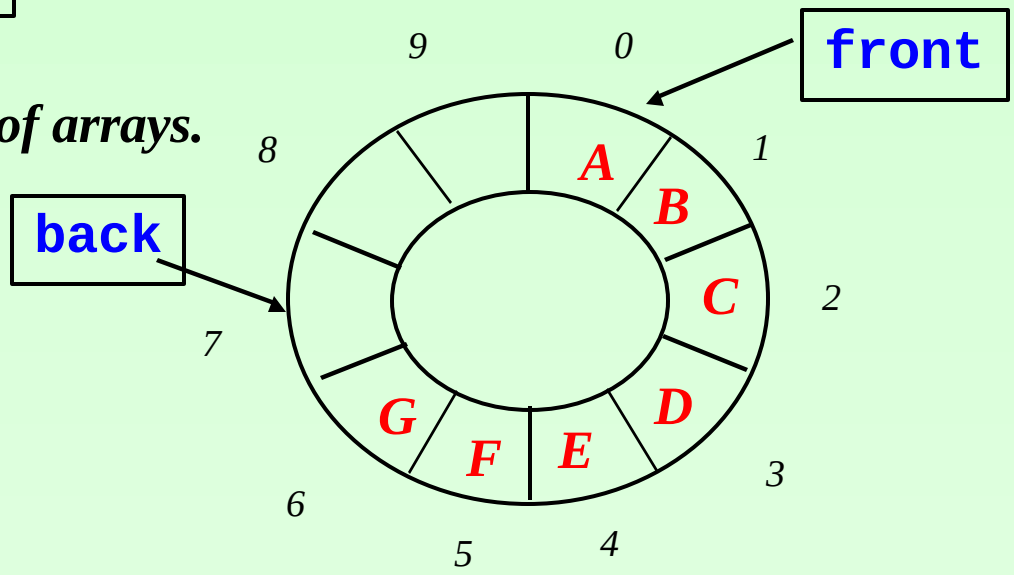


Circular Array

To implement queue, it is best to view arrays as circular structure.



Circular view of arrays.



How to Advance? Circular Array

Both front & back pointers should make advancement until they reach end of the array. Then, they should re-point to beginning of the array.

```
front = adv(front);
back  = adv(back);
```

```
public static int adv(int p)
 { int r = p+1;
   if (r<maxsize) return r;
   else return 0;
 }
```

upper bound of the array

Alternatively, use modular arithmetic:

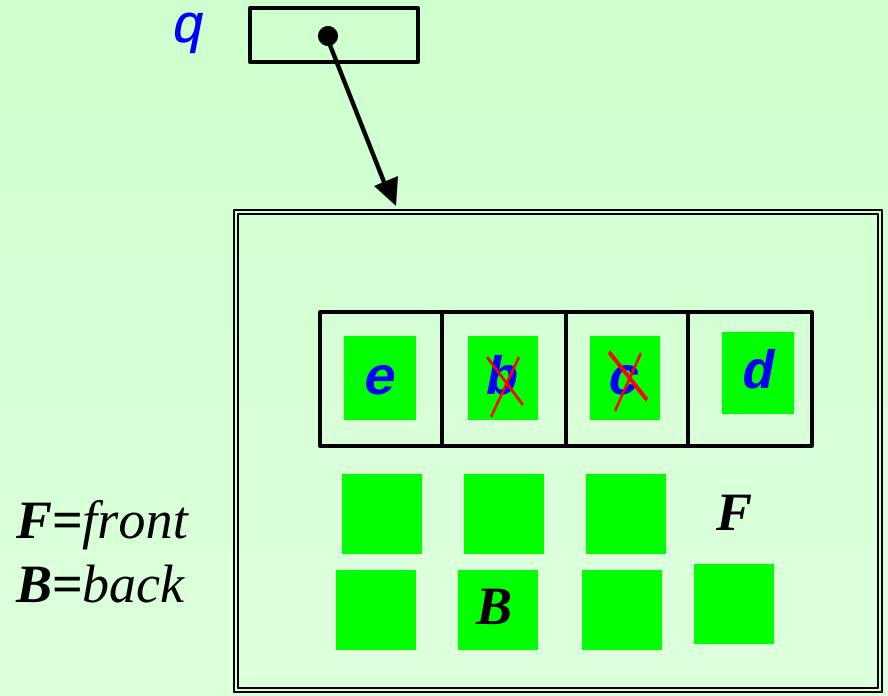
```
public static int adv(int p)
 { return ((p+1) % maxsize);
 }
```

mod operator

Sample

Circular Array

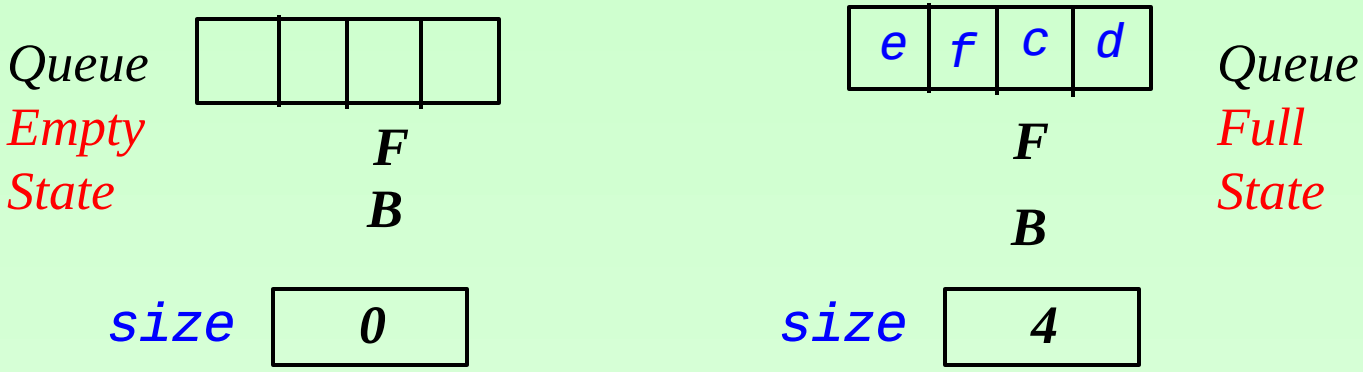
```
Queue q = QueueAR.makeQueue();
q.enqueue("a");
q.enqueue("b");
q.enqueue("c");
q.dequeue();
q.dequeue();
q.enqueue("d");
q.enqueue("e");
q.dequeue();
```



F=*front*
B=*back*

Checking for Full/Empty State — Circular Array

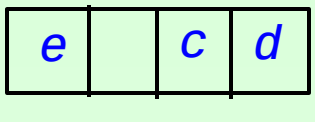
What does `(F==B)` *denote?*



Alternative - Leave a Deliberate Gap!

No need for `size` *field.*

Full Case : `(adv(B)==F)`



Codes — Implementation of Queue (Array)

```
class QueueAr implements Queue {

 private Object [] arr;
 private int front,back;
 private int maxSize;
 private final int initSize = 1000;
 private final int increment = 1000;

 public QueueAr() {arr = new Object[initSize]; front = 0; back=0; }

 public static Queue makeQueue() {return new QueueAr(); }

 public boolean isEmpty()          // check if queue is empty
  { return (front==back); }

 private boolean isFull()         // check if queue overflowing
  { return (adv(back)==front); }
  }
```

allows resizing of array

Codes for Queue Implementation — Circular Array

```
public void enqueue(Object o)    // add o to back of queue
 { if (this.isFull()) this.enlarge();
   arr[back]=o;
   back=adv(back); }

private void enlargeArr()        // enlarge the array
 {int newSize = maxSize+increment;
  Object [] barr = new Object[newSize];
  for (int j=0,k=front;j<=maxSize;j++,k=adv(k))
      {barr[j]=arr[k];};
  front=0; back=maxSize-1;
  maxSize = newSize; arr = barr;
 }
```

Codes for Queue Implementation — Circular Array

```
public void dequeue() throws Underflow
  { if this.isEmpty()
        {throw new Underflow("dequeue fails - empty q");}
    else front=adv(front);
  }


public Object getFront() throws Underflow
 { if this.isEmpty()
        {throw new Underflow("getFront fails - empty q");}
    else return arr[front];
 }
```