DIT181: Data Structures and Algorithms

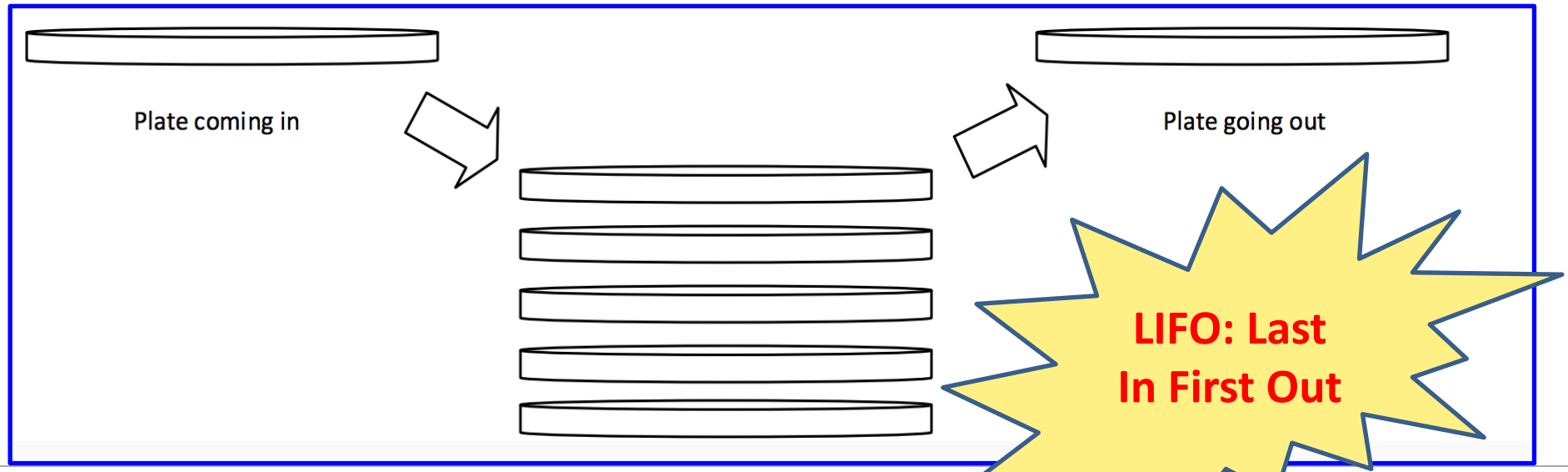
Stacks and Queues

Gül Calikli
Email: calikli@chalmers.se

- Kahoot questions

Stacks

- An abstract data type
- A good example of a ***stack*** is a ***stack of plates***.
- You cannot get the one on the bottom unless you pick up all the ones on the top of it.

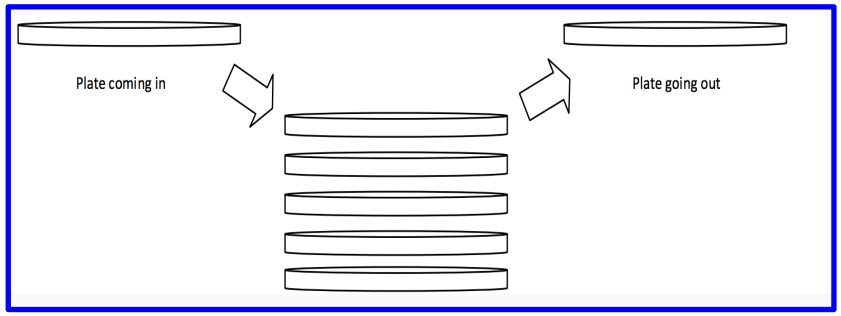


Queues

- Another common abstract data type is a queue.
- A real life example of a ***queue*** is a ***line of people waiting*** for some event. The first person in line will be served first, while the last person last.

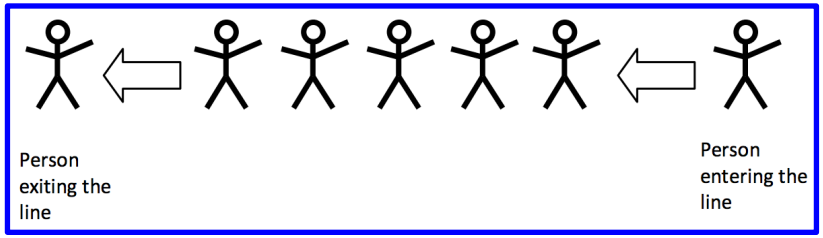
Stack vs. Queue

Stack



- We use:
  - ***push()*** to ***insert*** an element ***at the end***
  - ***pop()*** to ***remove*** the element ***at the end***

Queue



- We use:
  - ***enqueue()*** to ***insert*** an element ***at the end***
  - ***dequeue()*** to ***remove*** the element ***at the front***

Stack (Illustration)

- Set = $\{1, 2, 3\}$
- Stack = → | | →
  bottom top
- push(1)
- Set = $\{2, 3\}$
- **Stack = → | 1 | →**
- push(2)
- Set = $\{3\}$
- **Stack = → | 1 2 | →**
- push(3)
- Set = { }
- **Stack = → | 1 2 3 | →**
- pop()
- returns 3
- **Stack = → | 1 2 | →**
- pop()
- returns 2
- **Stack = → | 1 | →**
- pop()
- returns 1
- **Stack = → | | →**

Queue (Illustration)

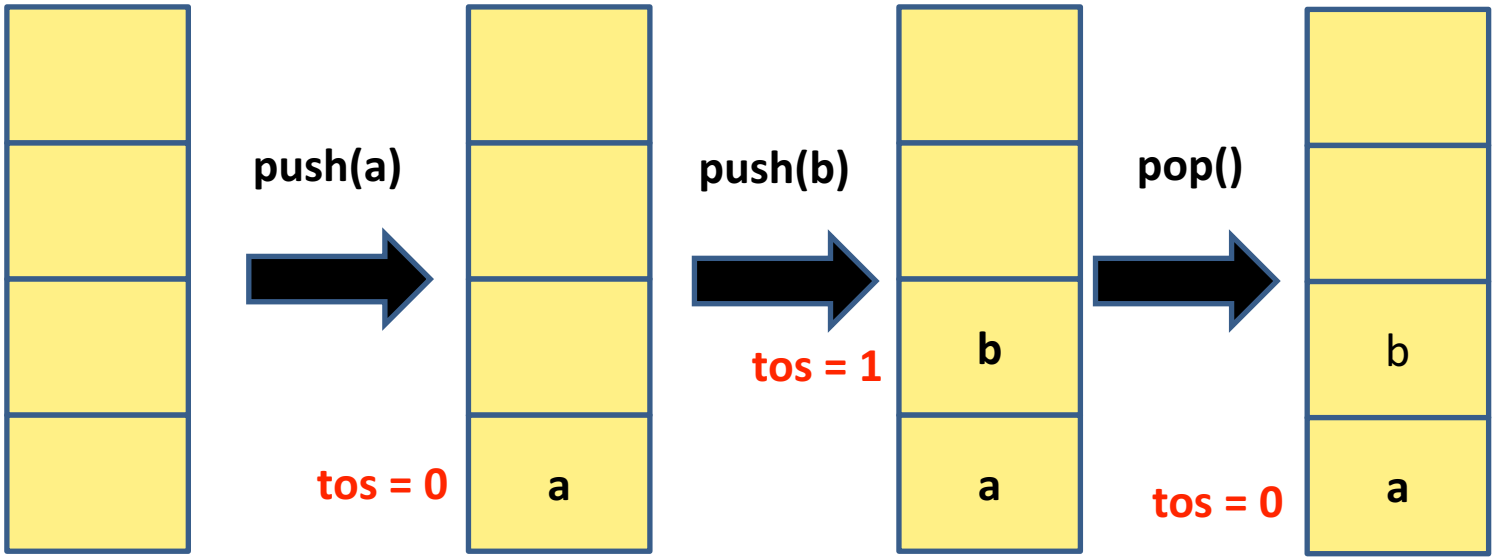
- Set = $\{1, 2, 3\}$
- Queue = ← | | ←
- enqueue(1)
- Set = $\{2, 3\}$
- **Queue = ← | 1 | ←**
- enqueue(2)
- Set = $\{3\}$
- **Queue = ← | 1 2 | ←**
- enqueue(3)
- Set = { }
- **Queue = ← | 1 2 3 | ←**
- dequeue()
- returns 1
- **Queue = ← | 2 3 | ←**
- dequeue()
- returns 2
- **Queue = ← | 3 | ←**
- dequeue()
- returns 3
- **Queue = ← | | ←**

Abstract Data Types

- You should distinguish between:
  - the abstract data type (ADT) (e.g., a stack, a queue)
  - its implementation (e.g. a dynamic array)
- Why?
  - When you use a data structure you don't care how it's implemented
  - Most ADTs have many possible implementations.
- See video

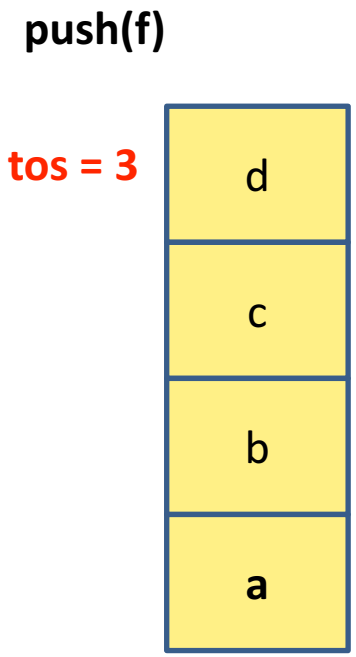
Stack Implementation

- A stack of characters can be implemented with an (dynamic) array and an integer.
  - The integer *top of stack* (*tos*) provides the array index of the top element of the stack



When **tos = -1**, the stack is **empty**

In-Class Exercise 5.1



- What is the complexity of push() operation in the worst case?
  - Hint: Think about what you should do, when array is full.
  - Further Hint: Double size of the array, when array is full.

Stack Implementation: Skeleton of the array based stack class

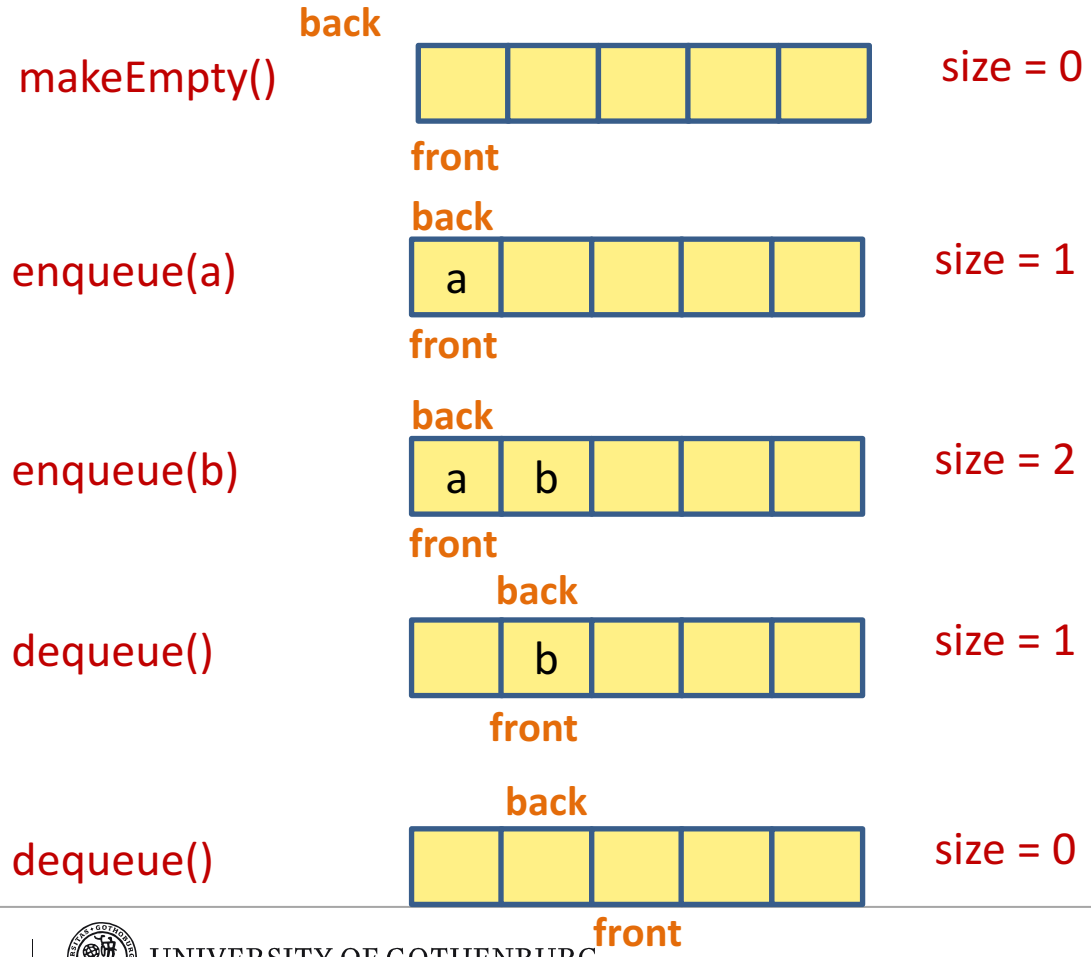
```
//ArrayStack class
//************ PUBLIC OPERATIONS ***************
// void push(x) --> Insert x
// void pop() --> Remove most recently inserted item
// AnyType top() --> Return most recently inserted item
// AnyType topAndPop() --> Return and remove most recent item
// boolean isEmpty() --> Return true if empty; else false
// void makeEmpty() --> Remove all items
// ************ ERRORS ***************
public class ArrayStack<AnyType> implements Stack<AnyType>
{       public ArrayStack() {  /* Construct the struct */
                theArray = (AnyType[]) new Object [DEFAULT_CAPACITY];
                topOfStack = -1; }
        public boolean isEmpty()  {  .........  }
        public void makeEmpty()    {  .........  }
        public AnyType top()        {  .........  }
        public void pop()            {  .........  }
        public AnyType topAndPop()  {  .........  }
        public void push(AnyType x)  {  .........  }
        private void doubleArray()  {  ........  }

        private AnyType [] theArray;
        private int topOfStack;

        private static final int DEFAULT_CAPACITY = 10;
}
```

In-Class Exercise 5.2: Implement the missing methods

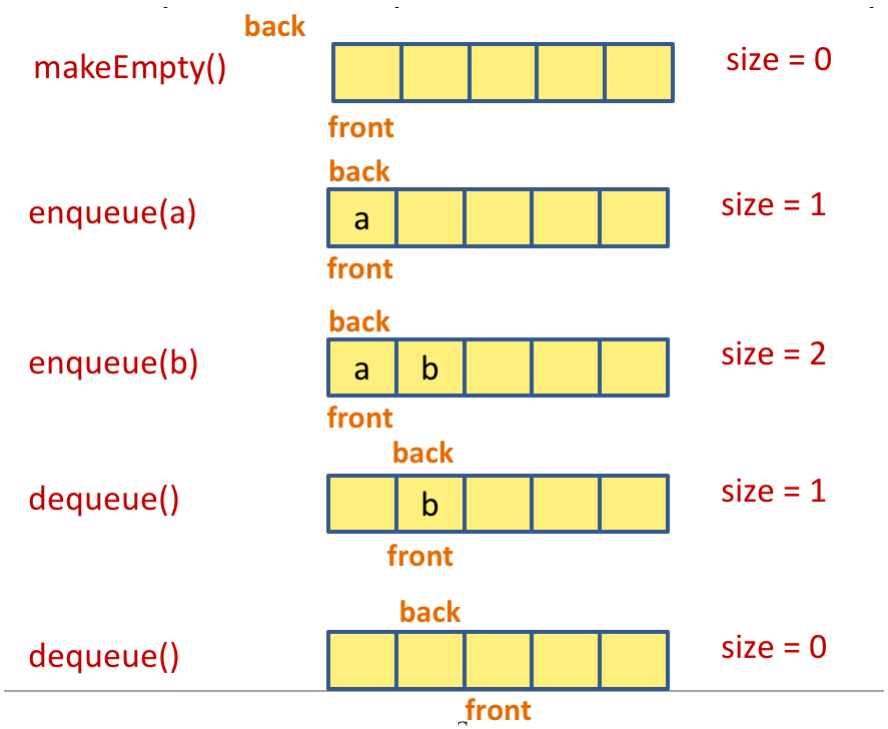
```
//ArrayStack class
//************ PUBLIC OPERATIONS ***************
// void push(x) --> Insert x
// void pop() --> Remove most recently inserted item
// AnyType top() --> Return most recently inserted item
// AnyType topAndPop() --> Return and remove most recent item
// boolean isEmpty() --> Return true if empty; else false
// void makeEmpty() --> Remove all items
// ************ ERRORS ***************
public class ArrayStack<AnyType> implements Stack<AnyType>
{       public ArrayStack() {  /* Construct the struct */
                theArray = (AnyType[]) new Object [DEFAULT_CAPACITY];
                topOfStack = -1; }
        public boolean isEmpty()  {  .........  }
        public void makeEmpty()    {  .........  }
        public AnyType top()       {  .........  }
        public void pop()          {  .........  }
        public AnyType topAndPop()  {  .........  }
        public void push(AnyType x)  {  .........  }
        private void doubleArray()  {  ........  }

        private AnyType [] theArray;
        private int topOfStack;

        private static final int DEFAULT_CAPACITY = 10;
}
```

Queue Implementation

- The easiest way to implement the queue is to store the items in an array with:
  - the front item in the front position (i.e., array index 0),
  - back represents the position of the last item in queue.



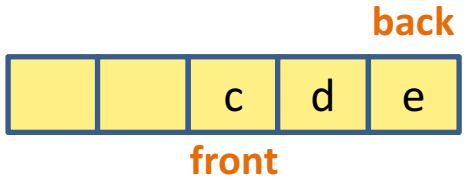
In-Class Exercise 5.3



- What is the time complexity of enqueue() implementation?
- What is the time complexity of dequeue() implementation?
- What is the fundamental problem with this approach?

Queue Implementation

After 3 enqueues

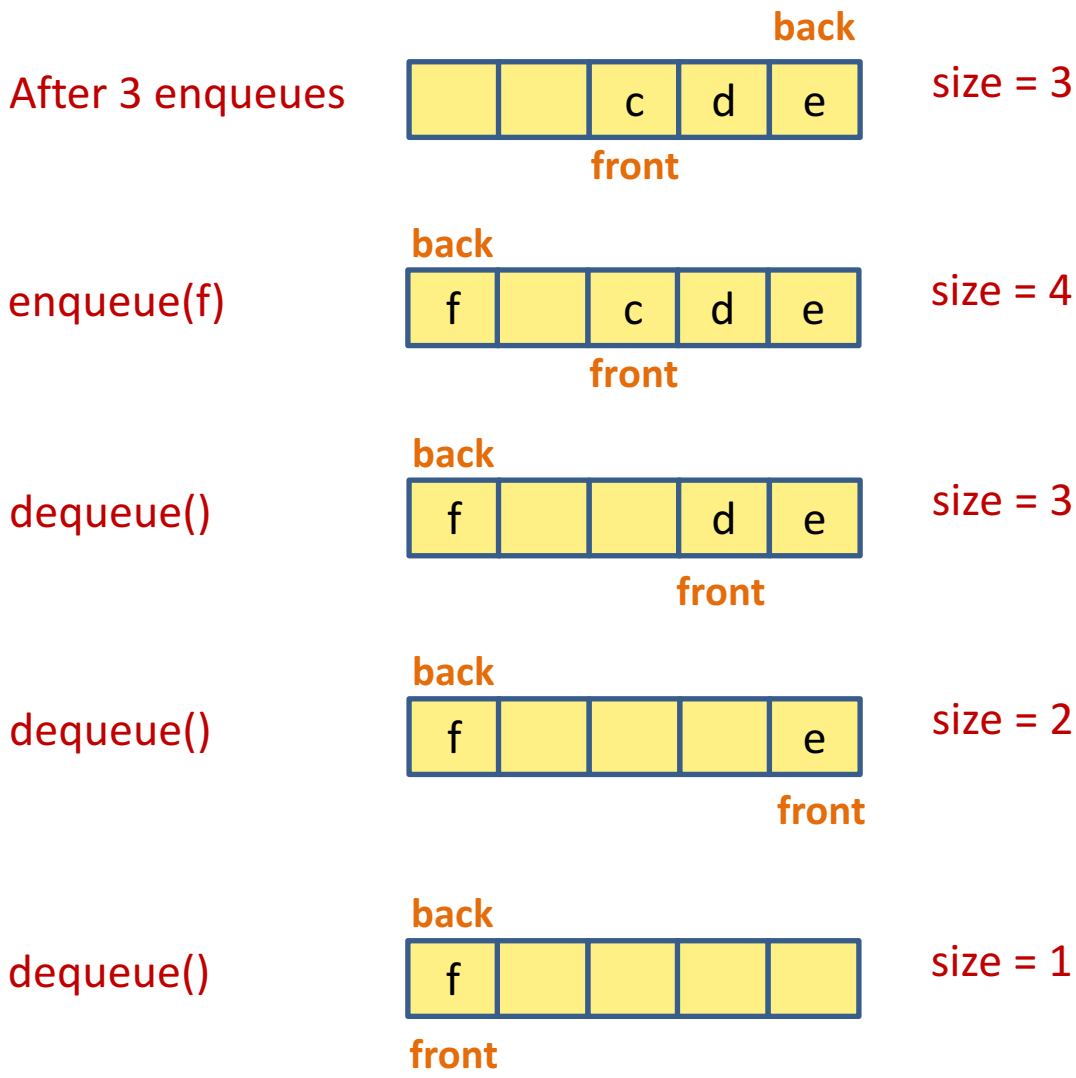


size = 3

- **Problem:** After 3 enqueue operations, we cannot add any more items, although the queue is not really full. There is plenty of extra space!!

- How can we solve the above problem?

Queue Implementation of the queue with the wraparound



Circular Array Implementation:

→ Returns **front** or **back** to the beginning of the array when either front or back reaches the end.

→ Double the size of the array only if the number of elements in the array is equal to the array size.

Queue Implementation: Skeleton of the array based queue class

```
//ArrayQueue class
//************ PUBLIC OPERATIONS ***************
// void enqueue(x) --> Insert x
// AnyType getFront() --> Return least recently inserted item
// AnyType dequeue() -->  Return and remove least recent item
// boolean isEmpty() --> Return true if empty; else false
// void makeEmpty() --> Remove all items
// ************ ERRORS ***************
public class ArrayQueue<AnyType>
{       public ArrayQueue() {  /* Construct the struct */
                theArray = (AnyType[]) new Object [DEFAULT_CAPACITY];
                makeEmpty(); }
        public boolean isEmpty()  {  .........  }
        public void makeEmpty()    {  .........  }
        public AnyType dequeue()           {  .........  }
        public AnyType getFront()           {  .........  }
        public void enqueue(AnyType x)  {  .........  }

        private int increment(x)  {  ........  }
        private void doubleQueue() {  ........  }
        private AnyType [] theArray;
        private int currentSize;
        private int front;
        private int back;
        private static final int DEFAULT_CAPACITY = 10; }
```

In-Class Exercise 5.4: Implement the missing methods

```
//ArrayQueue class
//************ PUBLIC OPERATIONS ***************
// void enqueue(x) --> Insert x
// AnyType getFront() --> Return least recently inserted item
// AnyType dequeue() -->  Return and remove least recent item
// boolean isEmpty() --> Return true if empty; else false
// void makeEmpty() --> Remove all items
// ************ ERRORS ***************
public class ArrayQueue<AnyType>
{       public ArrayQueue() {  /* Construct the struct */
                theArray = (AnyType[]) new Object [DEFAULT_CAPACITY];
                makeEmpty(); }
        public boolean isEmpty()  {  .........  }
        public void makeEmpty()    {  .........  }
        public AnyType dequeue()           {  .........  }
        public AnyType getFront()           {  .........  }
        public void enqueue(AnyType x)  {  .........  }

        private int increment(x)  {  ........  }
        private void doubleQueue() {  ........  }
        private AnyType [] theArray;
        private int currentSize;
        private int front;
        private int back;
        private static final int DEFAULT_CAPACITY = 10; }
```