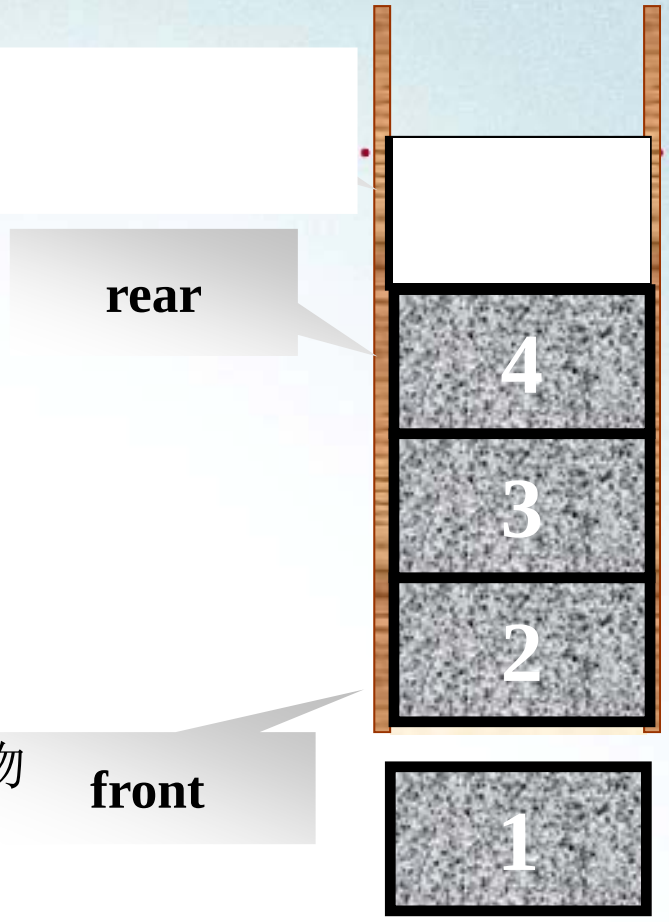
队列的顺序实现

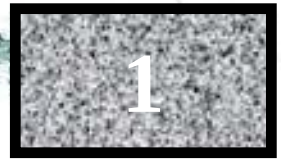
队列的顺序实现

- 基本思想：
  - 使用一个数组来存放队列中的元素，为了方便操作，还需保存队列中的队头和队尾位置
- 顺序实现：
  - 物理模型实现
  - 线性实现
  - 循环数组实现

队列的顺序实现

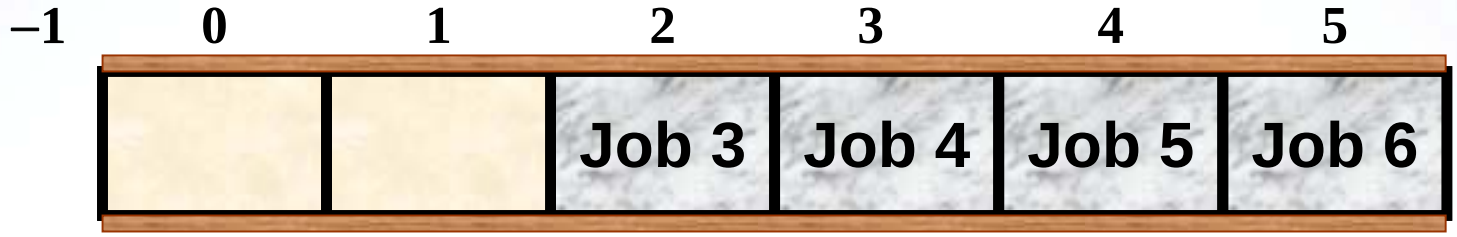
- 物理模型
  - 基本思想：队头始终位于数组首号元素位置
    - 入队时元素添加到最后，队尾指示器增1
    - 出队时，队列中所有元素都向前移动一个位置——逻辑上相邻，物理上也相邻
  - 不实用





队列的顺序实现

- 线性队列
  - 元素入队，队尾rear增加1，并将新元素放入该位置；
  - 元素出队，从队头front获取元素，并将front加1.



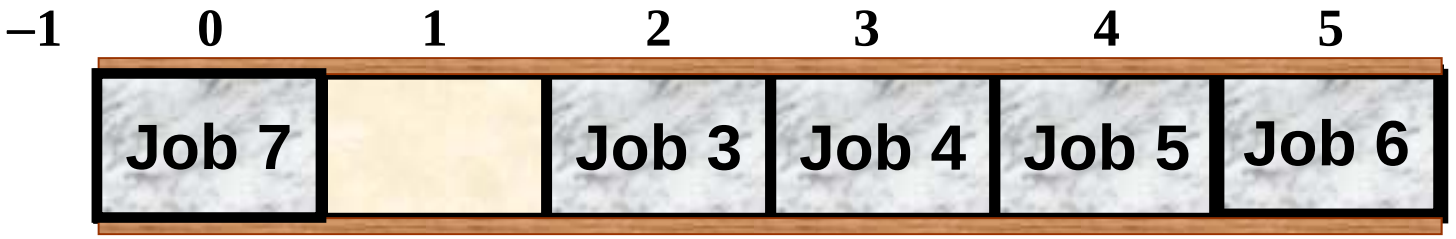
append Job 1	append Job 2	append Job 3
serve Job 1	append Job 4	append Job 5
append Job 6	serve Job 2	append Job 7

队列的顺序实现

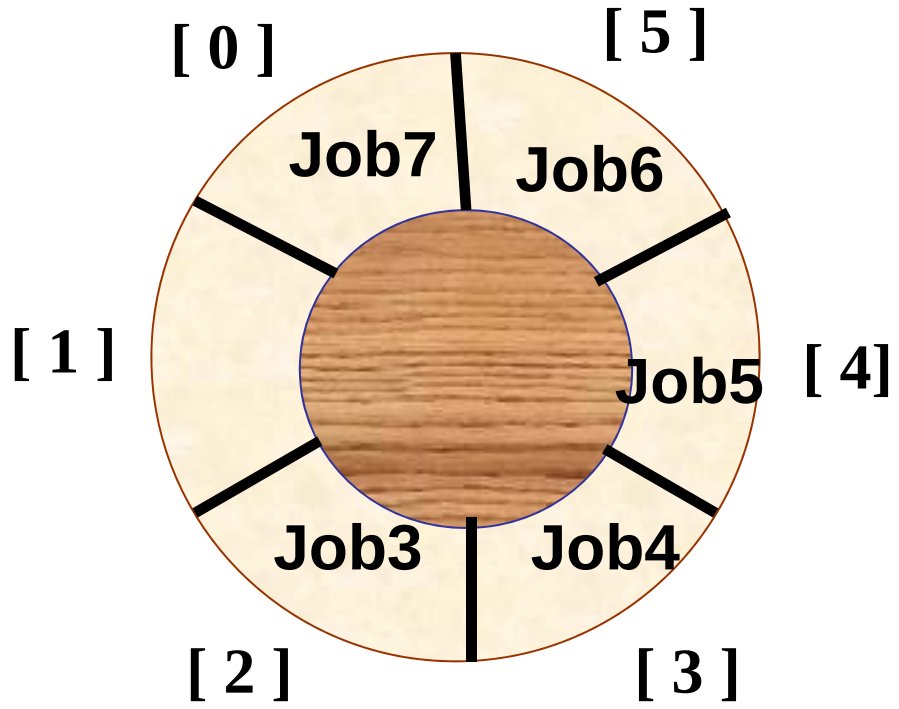
- 线性队列
  - 存在问题：
    - 假上溢
    - front和rear都是递增的，随着队列向数组后部移动，数组头部的存储空间将会被丢弃，无法再使用——空间使用效率低

队列的顺序实现

- 循环队列
  - 基本思想：不再将数组看成直线，而是一个圈（物理上线性，逻辑上循环）——克服假溢出的问题


–1
0
1
2
3
4
5
Job 7
Job 3
Job 4
Job 5
Job 6

append Job 7

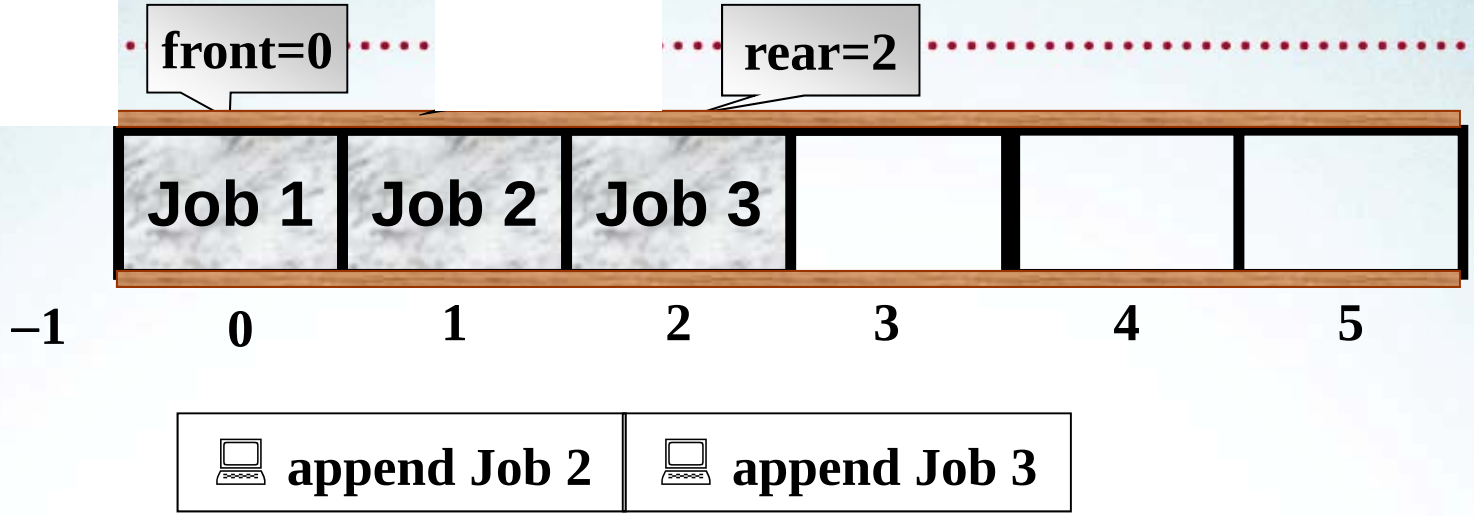

[0]
[5]
Job7
Job6
[1]
Job5
[4]
Job3
Job4
[2]
[3]

队列的顺序实现

❑循环队列的实现

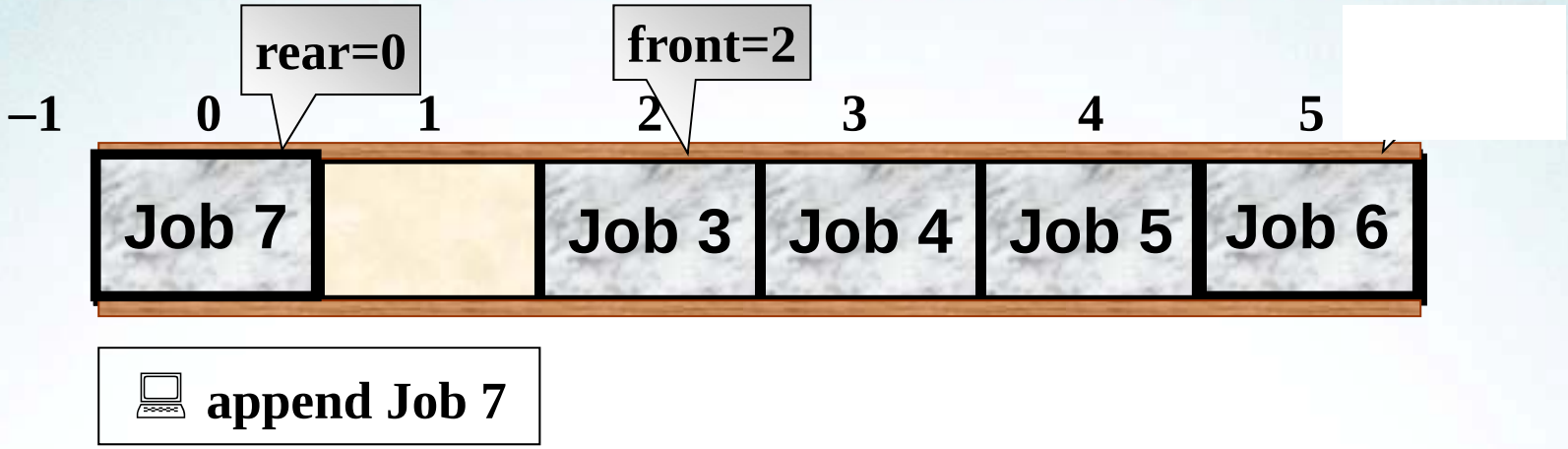
- 基本思想：数组被认为是首尾相连的
- front和rear分别指示着队头和队尾元素的位置
- 需要解决的问题：
  - 新元素如何入队？
  - 队头元素如何出队？
  - 初始化生成空队时，front和rear如何初始化？

循环队列中元素入队



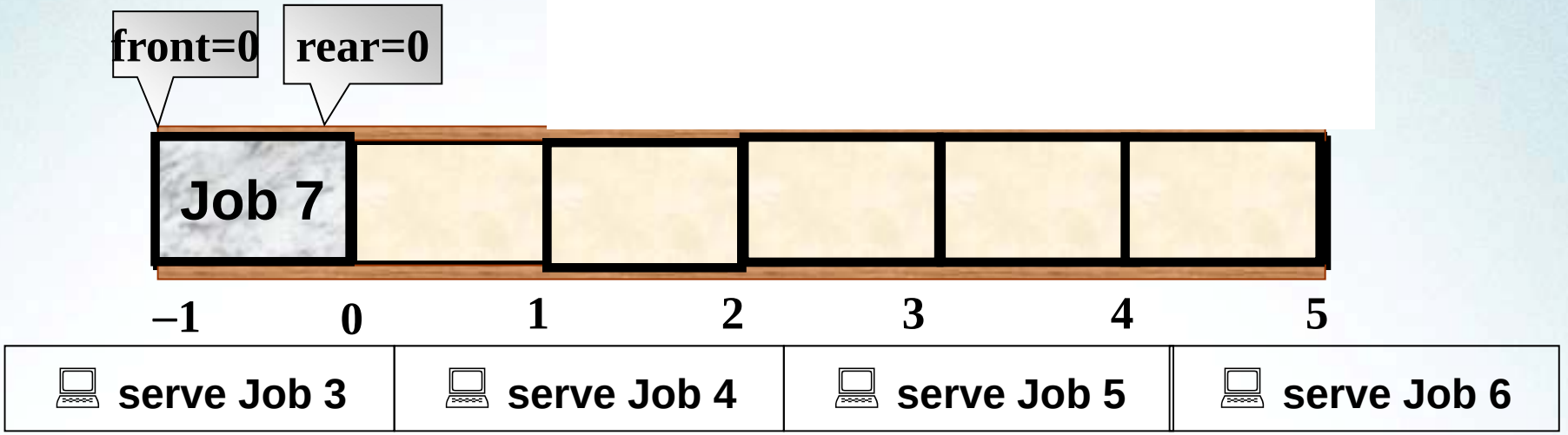
- 通常：元素入队时, rear++;即队尾指示器rear增1，front没有变化

循环队列中元素入队



- 但当rear移至数组的最后一个位置时(maxqueue-1)，rear=0;
- rear = ((rear + 1) ==maxqueue) ? 0 : (rear + 1);
- 或rear=(rear+1)%maxqueue;

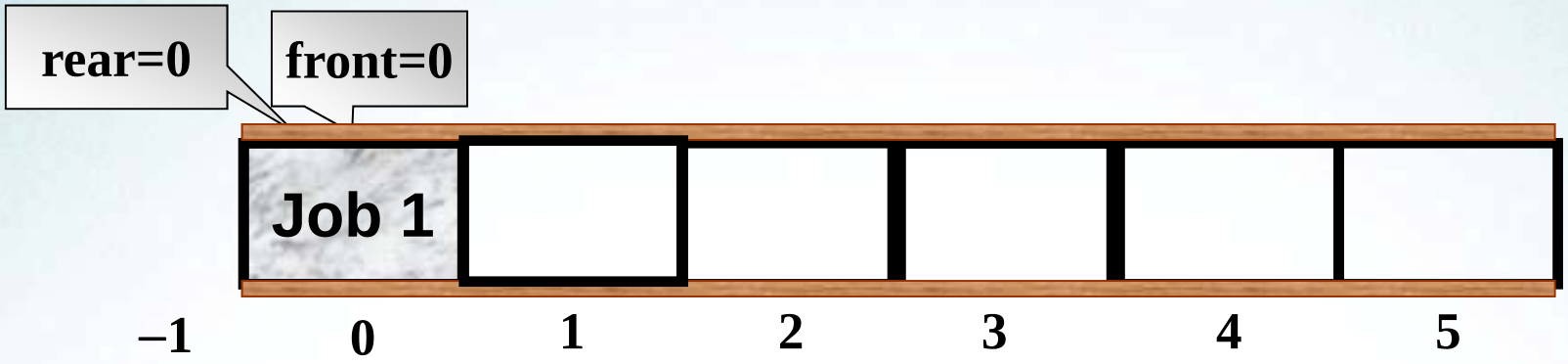
循环队列中元素出队



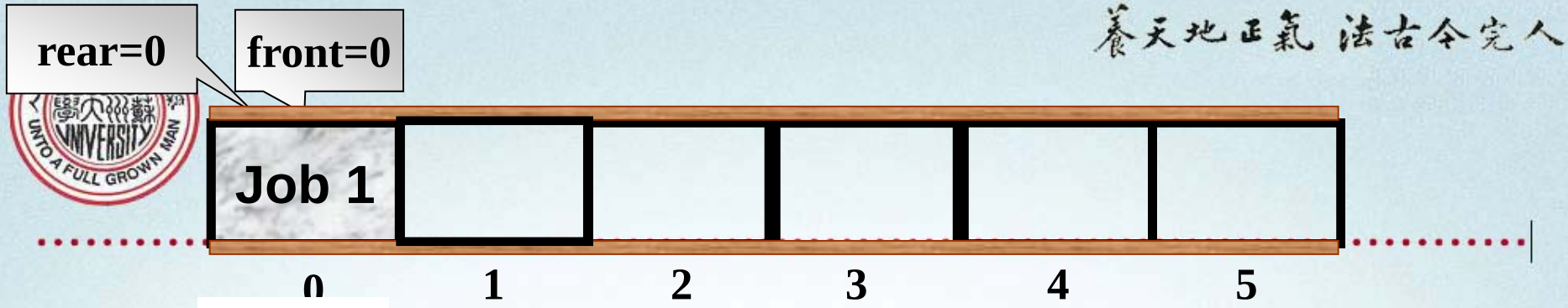
•通常，队列元素出队，front++;

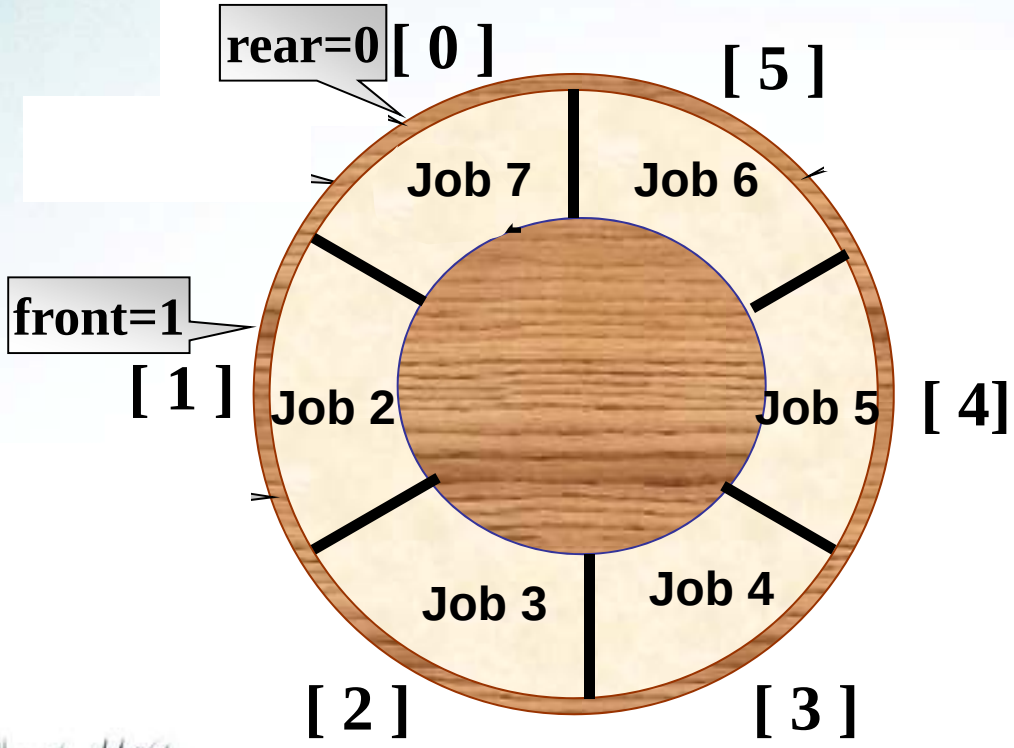
- 但front位于数组最后一个单元格时(maxqueue-1)，front=0;
- front = ((front + 1) ==maxqueue) ? 0 : (front + 1);
- 或 front=(front+1)%maxqueue;

循环队列中队头和队尾的初始化



- front=0;rear=-1; 或者
  front=0;rear=maxqueue-1;
- 经过一次入队后，可以得到这个状态。





队空时，rear和front相差1个位置。在这个例子中，front=1，rear=0；

队满时，rear和front也相差1个位置。在这个例子中，front=1，rear=0；

存在问题: 如何区别队空和队满!

sever Job 1 | append Job 2 | append Job 3，4，5，6

append Job 7

循环队列判断队满和队空

- 解决方法
  - 少用一个空间
    - 队列满——数组中仍然有一个单元空闲
  - 引入一个变量
    - 一个Boolean型变量表示rear是否刚刚到达front的前面；
    - 一个int型变量记录队列中当前元素的个数
    - 一个变量指示最后一次完成的动作是入队还是出队
  - ……

循环队列的实现

- 基本策略：
  - 队列存储在数组中
  - 借助额外的counter记录队列中元素的个数

循环队列的实现

```
const int maxqueue = 10; // small value for testing
class Queue {
public:
    Queue( );
    bool empty( ) const;
    Error_code serve( );
    Error_code append(const Queue_entry &item);
    Error_code retrieve(Queue_entry &item) const;
protected:
    int count;
    int front, rear;
    Queue_entry entry[maxqueue];
};
```

循环队列的实现

```
Queue :: Queue( )
/* Post: The Queue is initialized to be empty. */
{
    count = 0;
    rear = maxqueue - 1;
    front = 0;
}
bool Queue :: empty( ) const
/* Post: Return true if the Queue is empty, otherwise return false. */
{
    return count == 0;
}
```

循环队列的实现

```
Error_code Queue :: append(const Queue_entry &item)
/* Post: item is added to the rear of the Queue. If the Queue is full
return an Error_code of overflow and leave the Queue unchanged. */
{
        if (count >= maxqueue) return overflow;
        count++;
        rear = ((rear + 1) == maxqueue) ? 0 : (rear + 1);
        entry[rear] = item;
        return success;
}
```

循环队列的实现

```
Error_code Queue :: serve( )

/* Post: The front of the Queue is removed. If the Queue is empty return
an Error_code of underflow. */

{
        if (count <= 0) return underflow;

        count--;

        front = ((front + 1) == maxqueue) ? 0 : (front + 1);

        return success;

}
```

循环队列的实现

```
Error_code Queue :: retrieve(Queue_entry &item) const
/* Post: The front of the Queue retrieved to the output parameter
item. If the Queue is empty return an Error_code of underflow. */
{
        if (count <= 0) return underflow;
        item = entry[front];
        return success;
}
```

```
int Extended_queue :: size( ) const
/* Post: Return the number of entries in the Extended_queue. */
{
        return count;
}
```