一、栈与队列

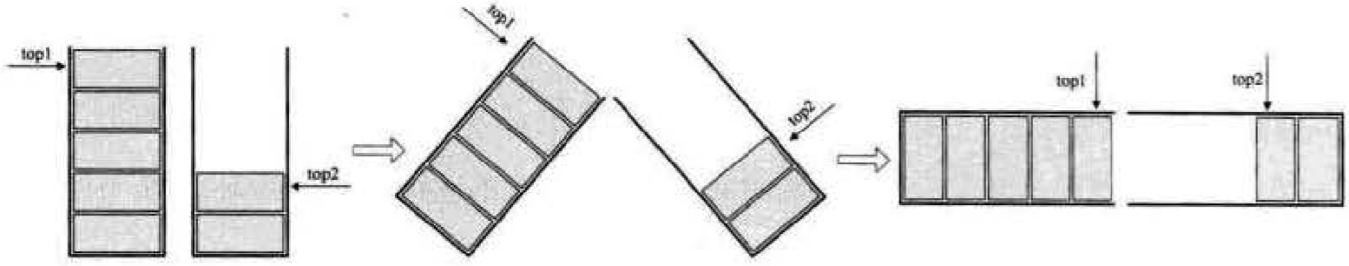
1. 栈

栈是限定仅在表尾进行插入和删除操作的线性表。

1-1. 栈的顺序存储结构

1-1-1. 两栈空闲空间

顺序栈的缺陷在于必须事先确定数组存储空间的大小，万一不够用，就需要使用编程手段来扩展数组的容量，非常麻烦。但对于两个相同类型的栈，我们将其合并扩展栈空间。
数组有两个端点，两个栈有两个栈底，让一个栈的栈底为数组的始端，即下标为0处，另一个栈为栈的末端，即下标为数组长度n-1处。这样，两个栈如果增加元素，就是两端点向中间延伸。



1-2. 栈的链式存储结构

1-3. 栈的作用

1-3-1. 递归

我们把一个把调用自己的函数称做递归函数。
迭代和递归的区别就是，迭代使用的是循环结构，递归使用的是选择结构。递归能使程序的结构更清晰，但是大量的递归调用会建立函数的副本，会耗费大量的时间和占用额外的内存；迭代则不需要反复调用函数和占用额外的内存。

1-3-2. 四则运算表达式

先说明下什么是后缀表达式
对于“ 9+ (3-1) * 3 + 10 % 2”,如果用后缀表示法应该是什么样子:"9 3 1 - 3 + 10 2 / +"这样的表达式称为后缀表达式，叫后缀的原因是在于**所有的符号都是要在运算数字的后面出现**

1-3-2-1. 后缀表达式计算结果

为了解释后缀表达式的好处，我们看计算机是如何应用后缀表达式计算出值
后缀表达式：9 3 1 - 3 + 10 2 / +
规则：从左到右编译表达式的每个数字和符号，遇到是数字就进栈，遇到是符号就弹出栈顶两个数字，进行运算，运算结果进栈，一直到最终获得结果

仍然存在一个问题：后缀表达式是如何推导出来的?即“ 9+ (3-1) * 3 + 10 % 2”，如何推出"9 3 1 - 3 + 10 2 / +"

1-3-2-2. 中缀表达式转后缀表达式

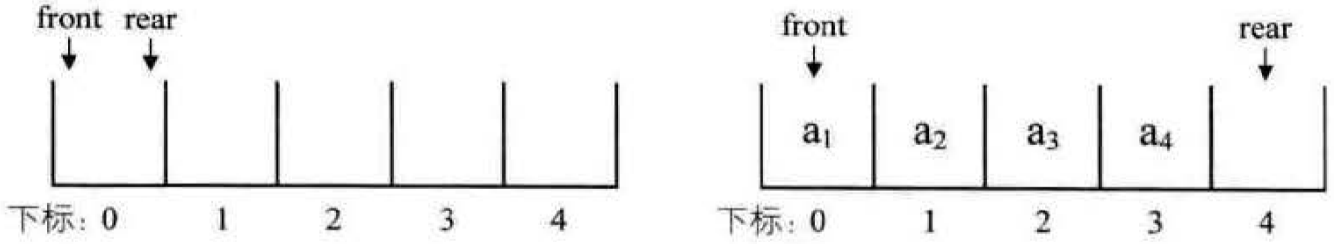
我们把平时所用的标准四则运算表达式“ 9+ (3-1) * 3 + 10 % 2”叫做中缀表达式，因为所有的运算符号都在两数字中间。
中缀表达式转后缀表达式
规则：从左到右遍历中缀表达式的每个数字和符号，若是数字就输出，即成为后缀表达式的一部分；若是符号，则判断其与栈顶符号的优先级，是右括号或优先级低于栈顶符号，则栈顶元素依次出栈并输出，并将当前符号进栈，直到最终输出后缀表达式为止

---

2. 队列

为了避免当只有一个元素时，队头和队尾重合使处理变得麻烦，所以引入两个指针，front指针指向队头元素，rear指针指向队尾元素的下一个位置，这样当front和rear相等时，队列不是还剩一个元素，而是空队列。

假设是长度为 5 的数组，初始状态，空队列如图 4-12-4 的左图所示，front 与 rear 指针均指向下标为 0 的位置。然后入队 a_1、a_2、a_3、a_4，front 指针依然指向下标为 0 位置，而 rear 指针指向下标为 4 的位置，如图 4-12-4 的右图所示。



出队 a_1、a_2，则 front 指针指向下标为 2 的位置，rear 不变，如图 4-12-5 的左图所示，再入队 a_5，此时 front 指针不变，rear 指针移动到数组之外。嗯？数组之外，那将是哪里？如图 4-12-5 的右图所示。

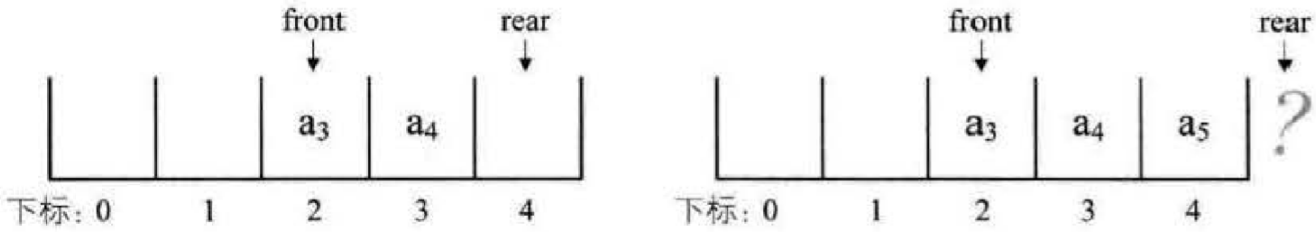
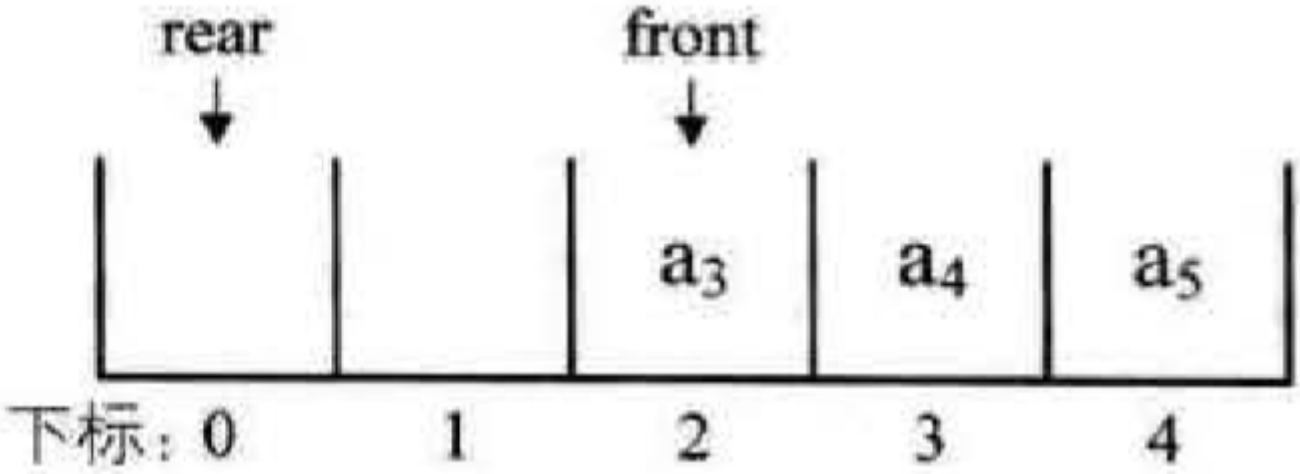


图 4-12-5

问题还不止于此。假设这个队列的总个数不超过 5 个，但目前如果接着入队的话，因数组末尾元素已经占用，再向后加，就会产生数组越界的错误，可实际上，我们的队列在下标为 0 和 1 的地方还是空闲的。我们把这种现象叫做“假溢出”。

2-1. 循环队列

为了解决假溢出问题，我们将头尾相接的顺序存储结构称为循环队列。
接下来就可以将上图的rear改为指向下标为0的位置：



接着入队 a_6，将它放置于下标为 0 处，rear 指针指向下标为 1 处，如图 4-12-7 的左图所示。若再入队 a_7，则 rear 指针就与 front 指针重合，同时指向下标为 2 的位置，如图 4-12-7 的右图所示。

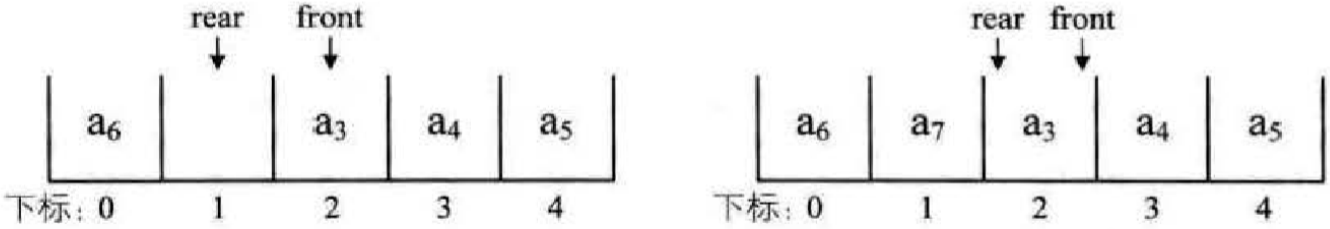
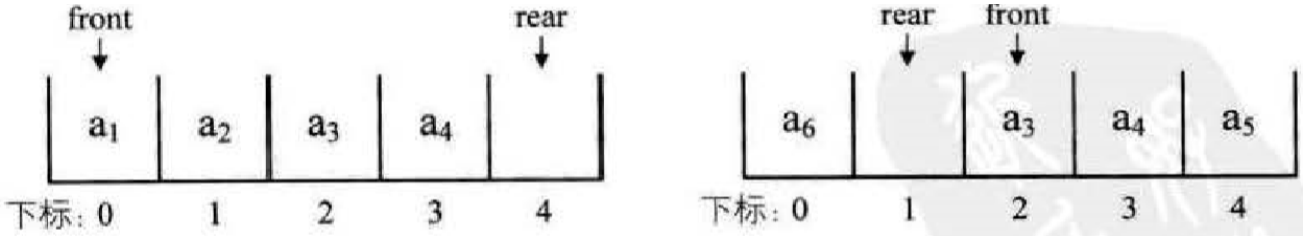


图 4-12-7

- 此时问题又出来了，我们刚才说，空队列时，front 等于 rear，现在当队列满时，也是 front 等于 rear，那么如何判断此时的队列究竟是空还是满呢？

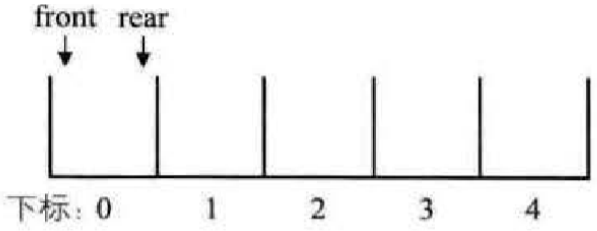
其中有两个解决方法：

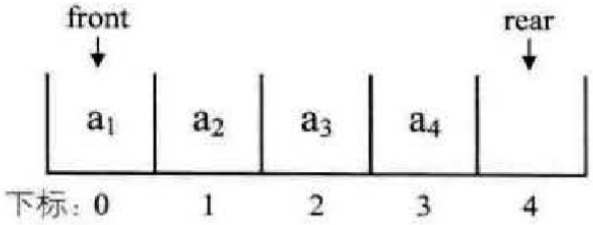
- 设置一个标志变量flag，当front rear，且flag = 0时为队列为空，当front rear，且flag=1时队列为满
- 另一个方法时当队列为空时，front==rear，当队列为满时，保留一个元素空间。如下图所示，我们就认为队列已满。也就是(rear+1)%QueueSize == front



2-1-1. 队列长度

- 当rear > front时，此时队列的长度为rear - front





- 当rear < front，此时队列的长度分为两半，左边为0+rear，右边是QueueSize-front
  队列长度为(rear-front+QueueSize)%QueueSize