3.15 将 splice 操作添加到 LinkedList 类中。该方法的声明

```
void splice(Iterator<T> itr, MyLinkedList<? extends T> lst )
```

将所有的项从 lst 中删除(使 lst 为空)，把它们放到 MyLinkedList this 中的 itr 之前，而 lst 和 this 必须是不同的表。你的程序必须以常数时间运行。

3.16 提供 ListIterator 的另一种方式是提供带有声明

```
Iterator<AnyType> reverseIterator( )
```

的表，它返回一个 Iterator，并被初始化至最后一项，其中 next 和 hasNext 被实现成与迭代器向表的前端(而不是向后)推进一致。然后，你可以通过使用程序

```
Iterator<AnyType> ritr = L.reverseIterator( );
while( ritr.hasNext( ) )
    System.out.println( ritr.next( ) );
```

反向打印 MyArrayList L。用这种思路实现 ArrayListReverseIterator 类，让 reverseIterator 返回一个新构造的 ArrayListReverseIterator。

3.17 修改 MyArrayList 类，通过使用在 3.5 节对 MyLinkedList 所看到的那些技巧以提供严格的迭代器检测。

3.18 对 MyLinkedList 类，通过分别调用私有的 add、remove、getNode 例程实现 addFirst、addLast、removeFirst、removeLast、getFirst 和 getLast 等方法。

3.19 不用头节点和尾节点重写 MyLinkedList 类，并描述该类和 3.5 节所提供的类之间的区别。

3.20 不同于我们已经给出的删除方法，另一种是使用**懒惰删除**(lazy deletion)的删除方法。要删除一个元素，我们只是标记上该元素被删除(使用一个附加的位(bit)域)。表中被删除和非被删除元素的个数作为数据结构的一部分被保留。如果被删除元素和非被删除元素一样多，则遍历整个表，对所有被标记的节点执行标准的删除算法。

a. 列出懒惰删除的优点和缺点。

b. 编写使用懒惰删除实现标准链表操作的相应例程。

3.21 用下列语言编写检测平衡符号的程序：

a. Pascal(begin/end,( ),[ ],{})

b. Java(/* */,( ),[],{ })

*c. 解释如何打印出一个很可能反映可能原因的错误信息。

3.22 编写一个程序计算后缀表达式的值。

3.23 a. 写出一个程序，将包含(,),+,-,* 和/等符号的中缀表达式转换成后缀表达式。

b. 将取幂运算符添加到指令系统中。

c. 编写一个程序将后缀表达式转换成中缀表达式。

3.24 编写只用一个数组而实现两个栈的例程。这些例程不应该声明溢出，除非数组中的每个单元都被使用。

3.25 *a. 提出一种数据结构支持栈 push 和 pop 操作以及第三种操作 findMin，它返回该数据结构中的最小元素。所有操作均以 $O(1)$ 最坏情形时间运行。

*b. 证明，如果我们添加找出并删除最小元素的第 4 种操作 deleteMin，那么至少有一种操作必然花费 $\Omega(\log N)$ 时间。(本题需要阅读第 7 章)

*3.26 指出如何用一个数组实现三个栈结构。

3.27 在 2.4 节中用于计算斐波那契数的递归例程如果对 $N=50$ 运行，栈空间有可能用完吗?