1 测试程序的设计思路

1. 我首先创建了一个链表 list1，测试一下默认的构造函数，即理应得到仅有 head 与 tail，size 为 0 的一个空链表。

2. 我再创建了一个 list2，通过 `List<int> list2 = {1, 2, 3, 4, 5};` 这种赋值构造的方法，理应得到 size 为 5 的一个链表。

3. 我再用 `List<int> list3(list2)` 这种方法测试拷贝构造函数，应该得到一个和 list2 一样的深度拷贝。

4. 接着再看移动构造函数是否正确，首先用 `std::move` 把刚才的 list2 转化为右值，形参列表中接受右值引用的移动构造函数被调用。移动后 list2 被接管，理应会被赋为空链表。

5. 接着测试接连赋值 $a = b = c$，以及自我赋值 $a = a$ 两种类型会不会导致错误。

6. **在非空链表中测试迭代器**，在此过程中，顺带测试**插入**。首先对空链表 list1 push 进一些元素，并打印出来，在打印的过程中也调用了 begin 与 end，即让 iterator 进行了前置以及后置 ++ 以及–运算的测试（共四种），其中，由于–it 的特殊性，在代码形式上稍有不对称的情况。

7. 测试 const 是否正确。顺带测试各种运算包括 == 与!= 运算符

8. 测试在指定位置进行 insert，`list1.insert(list1.insert(list1.begin()),temp);`，意在验证 insert 的返回值是否正确。接着用 `list1.insert(list1.begin(), 10+1);` 套用这种方法来检查在传入右值时的插入是否正确。

9. 测试 pop 以及 erase（对象**暂且都是非空的列表**）。看 erase 是否可在指定位置 erase，测试 erase 是否 from to 功能正常，并且返回值也是正确的，是否可以“套用”。

10. 测试 clear 清空列表，并且尝试 clear 以后再进行一次，验证对于空链表 clear 的效果。

11. 测试 int 之外的类别，比如 string，简单验证仍然成立。

12. 测试非 const 之下是否可以过迭代器进行修改。

2 测试的结果

使用 List.cpp 中的代码测试（**最后一部分暂且注释掉**），结果一切正常。输出如下：

```
List 1 size: 0 empty? 1
List 2 size: 5
List 3 size: 5
List 4 size: 5, List 2 size after move: 0
List 2's size becomes: 5 again
Now List2's elements are: 1 2 3 4 5
After list3=list3, list3's elements are: 1 2 3 4 5
Let's test push and pop
List1 front is : 30 back is : -5
List1 is (by ++i): 30 20 10 -5
List1 is (by i++): 30 20 10 -5
inverse List1 is (by --i): -5 10 20 30
```

```
inverse List1 is (by i--): -5 10 20 30
*const_it : 100
*Now const_it++ : 100
*Now ++const_it : 300
*Now --const_it : 200
*Now const_it-- : 200
*Whether equal? : 1
*Not equal? : 0
List1 after insert: 100 11 10 600 30 25 20 10 -5
List1 after pop 1st element, then erase 2nd element and pop the last element: 11 600 30 25 20 10
List1 after erase the last element: 11 600 30 25 20
After erase from to, list1 is : 11 20
After erase, the return iter points to: 20
List1 size after clear: 0
List1's content now :
StringList 1 size: 0
StringList 2 size: 4
StringList 2 content: Hello this is Maythics
StringList 2 content: Hello this is Maythics ' homepage
a a a a a
```

我用 `valgrind --leak-check=full ./List` 进行测试，发现没有发生内存泄露。

3 bug 报告

3.1 问题

将 List.cpp 最后的部分解注释，再次测试，会出现段错误等问题。

bug1（对迭代器未限制 ++ 与- -范围）:

1. 构建一个空列表
2. 调用 end 获取迭代器 iterator
3. 迭代器进行 ++ 操作，并且打印 *iterator

会出现段错误:

```
After illegal move:
Segmentation fault
```

原因是没有限制 iterator 的活动范围，应该在 ++ 与- -的函数处添加判断，使得 iterator 到达不了哨兵节点以外的地方（但是哨兵节点要允许到达，因为如循环打印出整个 list 的功能就需要 iterator 能够到达哨兵处比较）

bug2（erase 一个空链表），触发如下:

1. 构建一个空列表
2. 调用 begin 获取迭代器

3. 调用 erase 删除该迭代器所指位置

会出现段错误:

```
Try to erase an empty list:
Segmentation fault
```

它出现的原因是：erase 函数本身并未判断是否链表为空，如果真的为空，理应不做任何操作直接 return，但是此时却尝试删除而出错。

3.2 解决方案

将 `#include "List.h"` 注释掉，并且改成 `#include "myList.h"`，即可解决问题。