1 测试程序的设计思路

1. 首先在List.h文件中我补充了find(const Object &val)函数和resize()函数，其中find()函数输入要查找的值，返回第一个符合条件的值所在的位置，若 val 不在链表里返回-1。resize()函数第一个参数输入 resize 后链表的长度 n；如果 $n < theSize$，将链表多余的元素直接去掉；如果 $n > theSize$, 将链表多余部分用第二个参数给出的元素填充。我补充了print()函数打印列表，便于观察结果。最后为了清楚的看到析构函数的调用，我修改了析构函数并且希望每次析构会输出the process is constructing.

2. 为了测试构造函数，我先构造了空链表l1，测试默认构造函数，打印l1并利用push_front()函数向其中依次添加元素 0 到 4，我们希望此时l1应该形如4 3 2 1 0，之后测试右值引用的push_front()函数，利用push_front(10+0)向l1添加元素，之后打印l1观察结果。为了测试动态的front()函数，我利用l1.front()=1，将l1的第一个元素改为了 1，然后用l1.pop_back()函数删除了l1的最后一个元素，再打印l1观察与之前的异同。

3. 为了测试拷贝构造函数，构造了List<int>l6=l1，利用push_back()函数向其中添加了元素 0 到 4，这时候我希望l4应该形如0 1 2 3 4，为了测试右值引用的push_back函数，我通过l6.push_back(0+10)，l6.push_back(1+10)添加了 10 和 11 两个元素，打印观察。之后为了测试动态的back()函数，我用l6.back()=0，将l6的最后一个元素修改为 0，之后测试pop_front()函数删除l6的第一个元素，打印l6观察结果。

4. 我创建了List<double>l2={1.2,1.3，1.4，1.5}测试列表的初始化功能，又利用const List<double>l3=l2构造了l2用来测试静态的front()back()函数，打印结果。

5. 为了测试移动构造函数，我构造了List<int>l，并向其中添加了元素0 1 2, 之后利用List<int>l5(move(l))将结果移动给l5，之后我们输出l观察是否发生了复制，结果显示l变成了空链表，没有复制，之后利用l5测试empty()，size()函数，然后打印l5，利用l5.resize(6,1)测试resize()函数，再利用l5.find(10)，l5.find(1)测试find()函数。打印。

6. 利用两个iterator itr1 itr2测试左值引用和右值引用的insert()erase()函数。

7. 最后我构造了空链表l8测试begin()end()函数并向其中添加了一个元素，是为了测试不同情况下的begin()end()函数。

2 测试的结果

测试结果一切正常。

我用 valgrind 进行测试，发现没有发生内存泄露。最终在终端输出的结果如下：

```
the list is empty
11 10 4 3 2 1 0
1 10 4 3 2 1
0 1 2 3 4 10 11
1 2 3 4 10 0
the first element in l3 is 1.2
the last elemnet in l3 is 1.5
1.2 1.3 1.4 1.5
the process is constructing
1 2 3
the list is empty
show that if l5 is empty 0
```

```
the size of l5 is 3
0 1 2
0 1 2 1 1 1
to check if 10 is in l5-1
to check if 1 is in l51
1.2 0.03 1.3 0.04 1.4
1.2 0.03 1.3 0 0.04 1.4
the list is empty
the list is empty
Single element: 10
the process is constructing
the process is constructing
the process is constructing
the process is constructing
the process is constructing
the process is constructing
the process is constructing
the process is constructing
==6127==
==6127== HEAP SUMMARY:
==6127==     in use at exit: 0 bytes in 0 blocks
==6127==   total heap usage: 59 allocs, 59 frees, 75,096 bytes allocated
==6127==
==6127== All heap blocks were freed -- no leaks are possible
==6127==
==6127== For lists of detected and suppressed errors, rerun with: -s
==6127== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```