1 测试程序的设计思路

本测试程序采用模块化设计，将测试分为多个独立的测试单元，每个单元负责测试 List 类的不同功能方面。主要测试模块包括：

1. **构造函数和析构函数测试（testConstructorsAndDestructors）**
   - 测试默认构造函数
   - 测试初始化列表构造
   - 测试拷贝构造
   - 测试移动构造
2. **迭代器功能测试（testIterators）**
   - 测试 begin() 和 end()
   - 测试迭代器自增操作
   - 测试 const 迭代器
   - 测试范围遍历
3. **插入和删除操作测试（testInsertionAndDeletion）**
   - 测试 push_back 和 push_front
   - 测试 insert
   - 测试 pop_front 和 pop_back
   - 测试 erase
   - 测试 clear
4. **赋值操作测试（testAssignment）**
   - 测试拷贝赋值
   - 测试移动赋值

为了更全面地测试 List 类的功能，我特别设计了 TestObject 类，这个类具有完整的移动语义支持，用于测试 List 在处理复杂对象时的行为。同时，每个测试函数都使用 assert 语句进行结果验证，确保功能正确性。

2 测试的结果

所有测试模块均通过测试，具体结果如下：

1. 构造函数和析构函数测试通过，验证了：
   - 空列表的正确构造
   - 初始化列表构造的正确性
   - 拷贝构造的深复制特性
   - 移动构造后源对象的状态
2. 迭代器功能完全正常，包括：

- 正向遍历
- 迭代器自增操作
- const 迭代器的只读特性

3. 插入和删除操作测试结果显示：
   - 所有插入操作保持了列表的完整性
   - 删除操作正确处理了节点的链接
   - clear 操作完全清空了列表
4. 赋值操作测试证实：
   - 拷贝赋值实现了深复制
   - 移动赋值正确转移了资源所有权
5. 递减运算符测试结果显示：
   - 前置递减（–iterator）正确修改迭代器位置并返回引用
   - 后置递减（iterator–）正确保存原值并返回副本
   - const 迭代器的递减操作保持了只读特性
   - 在链表边界进行递减操作时行为正确

使用 valgrind 进行内存泄漏检测，结果显示没有内存泄漏：

```
==1234== All heap blocks were freed -- no leaks are possible
```

3 Bug 报告

在测试过程中，我发现了一个潜在的安全性问题：

1. List 类的 front() 和 back() 方法在空列表上的行为未定义
2. 当 list 为空时调用这些方法会导致未定义行为
3. 在实际应用中可能导致程序崩溃

建议的改进方案：

Listing 1: 改进后的 front() 方法示例

```
Object& front() {
    if (empty()) {
        throw std::runtime_error("Accessing front() on empty list");
    }
    return *begin();
}
```

这个 bug 的根本原因是缺少边界检查。在双向链表实现中，虽然使用了头尾哨兵节点，但这些方法仍然假设列表非空。在健壮的实现中，应该添加适当的错误处理机制。