1 简介

在本次测试中，我创建了一个自定义链表类，涵盖了整数、浮点数和字符类型的链表。测试过程中，我向链表中插入了多种元素，并进行了删除、访问、插入、清空和异常处理等操作，以验证链表的功能和稳定性。

2 测试程序的设计思路

链表初始化：首先，我实例化了不同类型的链表对象，包括 List<int>、List<float> 和 List<char>。对每个链表对象进行了初始化并确认其初始状态为“空”。

元素插入：对每个链表使用 pushback 和 pushfront 方法插入元素。例如，在字符链表中，我按顺序插入了字母“wanghengning”，以确保插入操作的顺序和准确性。

访问和修改操作：

通过调用 front() 和 back() 方法，检查链表的首尾元素，并验证其正确性。使用迭代器遍历链表中的元素，确保可以按预期访问所有元素。删除操作：对链表进行 popfront 和 popback 操作，以测试元素的删除功能。尤其是对字符链表的操作，删除后验证链表大小和元素状态。

异常处理：在空链表上调用 front() 和 back() 方法，捕捉异常，确保程序能正确处理非法操作。

拷贝和移动构造：

对链表进行拷贝构造和移动构造，测试在边界情况下的表现，例如从空链表进行拷贝或移动，检查目标链表的状态。确认拷贝构造后的链表大小和内容与源链表一致，移动构造后源链表应为空。范围删除：在字符链表上进行范围删除，验证清空列表的操作是否成功，并检查链表状态是否符合预期。

嵌套链表：创建一个外层链表 List<List<int»，并向其中插入内层链表，测试嵌套结构的操作，包括插入、访问和删除等。

3 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。

4 （可选）bug 报告

我发现了一个 bug，触发条件如下：

1. 首先……
2. 然后……
3. 此时发现……

据我分析，它出现的原因是：