计算机科学技术学院实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名称** | 面向对象程序设计 | | | **学 号** |  |
| **实验项目** | 实验三 | | | **姓 名** |  |
| **学 时** | 2 | **项目性质** | 综合型 | **班 级** | 2405111B |
| **指导教师** | 蒋振刚 | **实验地点** | 实训楼424 | **日 期** | 2025年4月7日 |
| 1. **实验目的和要求**   第3个实验：为用链表实现的线性表添加赋值和拷贝。  目的：了解深拷贝和浅拷贝的区别。  要求：在线性表类DList的基础之上，扩充：  1）拷贝构造函数；  2）赋值运算；  3）测试代码验证 以上的操作。 | | | | | |
| 1. **实验环境**   系统：fedora 41  环境：g++ | | | | | |
| 1. **实验内容与过程**   **上一次的Dlist.h:**  **#ifndef DLIST\_H**  **#define DLIST\_H**  **#include <iostream>**  **class DList {**  **public:**  **// 构造函数**  **DList();**    **// 析构函数**  **~DList();**    **// 清空线性表**  **void clear();**    **// 判断线性表是否为空**  **bool isEmpty() const;**    **// 获取线性表长度**  **int size() const;**    **// 获取指定位置的元素**  **bool getElem(int i, int &e) const;**    **// 查找元素位置（返回第一次出现的位置，未找到返回0）**  **int locate(int e) const;**    **// 在指定位置插入元素**  **bool insert(int i, int e);**    **// 删除指定位置的元素**  **bool remove(int i, int &e);**    **// 打印线性表**  **void display() const;**  **protected:**  **struct Node {**  **int data;**  **Node \*prev;**  **Node \*next;**  **};**  **Node \*head;**  **Node \*tail;**  **int length;**  **};**  **DList::DList() : length(0) {**  **head = new Node;**  **tail = new Node;**  **head->next = tail;**  **tail->prev = head;**  **}**  **DList::~DList() {**  **clear();**  **delete head;**  **delete tail;**  **}**  **void DList::clear() {**  **Node \*p = head->next;**  **while (p != tail) {**  **Node \*q = p;**  **p = p->next;**  **delete q;**  **}**  **head->next = tail;**  **tail->prev = head;**  **length = 0;**  **}**  **bool DList::isEmpty() const {**  **return length == 0;**  **}**  **int DList::size() const {**  **return length;**  **}**  **bool DList::getElem(int i, int &e) const {**  **if (i < 1 || i > length) {**  **std::cout << "位置无效！" << std::endl;**  **return false;**  **}**  **Node \*p = head->next;**  **for (int j = 1; j < i; j++) {**  **p = p->next;**  **}**  **e = p->data;**  **return true;**  **}**  **int DList::locate(int e) const {**  **Node \*p = head->next;**  **for (int i = 1; p != tail; i++) {**  **if (p->data == e) {**  **return i;**  **}**  **p = p->next;**  **}**  **return 0;**  **}**  **bool DList::insert(int i, int e) {**  **if (length >= 100) {**  **std::cout << "线性表已满，无法插入！" << std::endl;**  **return false;**  **}**  **if (i < 1 || i > length + 1) {**  **std::cout << "插入位置无效！" << std::endl;**  **return false;**  **}**  **Node \*p = head;**  **for (int j = 0; j < i; j++) {**  **p = p->next;**  **}**  **Node \*q = new Node;**  **q->data = e;**  **q->prev = p->prev;**  **q->next = p;**  **p->prev->next = q;**  **p->prev = q;**  **length++;**  **return true;**  **}**  **bool DList::remove(int i, int &e) {**  **if (i < 1 || i > length) {**  **std::cout << "位置无效！" << std::endl;**  **return false;**  **}**  **Node \*p = head->next;**  **for (int j = 1; j < i; j++) {**  **p = p->next;**  **}**  **e = p->data;**  **p->prev->next = p->next;**  **p->next->prev = p->prev;**  **delete p;**  **length--;**  **return true;**  **}**  **void DList::display() const {**  **Node \*p = head->next;**  **while (p != tail) {**  **std::cout << p->data << " ";**  **p = p->next;    }**  **std::cout << std::endl;**  **}**    **#endif**  **这一次的:**  **DlistPlus.h**  **#ifndef DLIST\_PLUS\_H**  **#define DLIST\_PLUS\_H**  **#include "/work2/include/Dlist.h"**  **//如需使用自行修改路径**  **class DList\_plus:public DList{**  **public:**  **using DList::DList;**  **~DList\_plus() {}**  **// 拷贝构造函数**  **DList\_plus(const DList\_plus& other);**  **// 赋值运算符重载**  **DList\_plus& operator=(const DList\_plus& other);**  **};**  **// 拷贝构造函数实现**  **DList\_plus::DList\_plus(const DList\_plus& other) : DList() {**  **// 首先确保当前链表为空**  **clear();**    **// 遍历other链表，复制每个节点**  **Node\* current = other.head->next;**  **while (current != other.tail) {**  **insert(length + 1, current->data); // 在末尾添加元素，位置是当前长度+1**  **current = current->next;**  **}**  **}**  **// 赋值运算符重载实现**  **DList\_plus& DList\_plus::operator=(const DList\_plus& other) {**  **// 自我赋值检查**  **if (this == &other) {**  **return \*this;**  **}**    **// 清空当前链表**  **clear();**    **// 复制other链表的所有节点**  **Node\* current = other.head->next;**  **while (current != other.tail) {**  **insert(length + 1, current->data); // 在末尾添加元素，位置是当前长度+1**  **current = current->next;**  **}**    **return \*this;**  **}**  **#endif**  **Main.cpp:**  **#include <iostream>**  **#include "include/DlistPlus.h"**  **using namespace std;**  **int main(){**  **DList\_plus list;**  **int e;**    **std::cout << "测试线性表基本操作：" << std::endl;**    **// 测试插入操作**  **std::cout << "\n===== 测试插入操作 =====" << std::endl;**  **list.insert(1, 10);**  **list.insert(2, 20);**  **list.insert(3, 30);**  **list.insert(2, 15); // 在中间插入**  **list.display();**    **// 赋值运算符重载**  **std::cout << "\n===== 赋值运算符重载 =====" << std::endl;**  **DList\_plus list2;**  **list2 = list;**  **list2.display();**    **return 0;**  **}**  **Dlist.h部分的:**  wps  DlistPlus.h部分的: | | | | | |
| 1. **实验结果与分析**   **电脑萤幕的截图  AI 生成的内容可能不正确。**  DList\_plus 是对已有双向链表类 DList 的功能增强，继承了其所有基本操作，并特别实现了拷贝构造函数和赋值运算符重载，从而支持链表对象之间的深拷贝。通常在使用链表类时，若未实现这些操作，默认的拷贝只会复制指针地址，导致多个对象指向同一片内存区域，可能引发悬空指针、重复释放等问题。而 DList\_plus 通过显式实现拷贝构造和赋值重载，确保每次复制操作都会重新创建链表节点，使新对象拥有独立完整的数据结构。在实现中，程序会先清空当前链表，再从源对象中逐个读取数据并插入到当前链表末尾，使用 insert(length + 1, data) 保证顺序一致。 | | | | | |
| 1. **实验心得**   通过本次实验，我深入理解了深拷贝和浅拷贝的区别，并掌握了如何在使用链表实现的线性表类中正确添加拷贝构造函数和赋值运算符重载。在继承自基础类 DList 的子类 DList\_plus 中，我编写了拷贝构造函数和 operator=，实现了对链表节点的逐个复制，确保每个对象在复制后都有自己独立的一份数据。  实验过程中，我意识到如果未正确实现深拷贝，多个对象会共享同一块内存，一旦一个对象析构或修改数据，可能会影响另一个对象，导致程序异常甚至崩溃。而深拷贝通过为新对象重新分配内存并复制内容，避免了这种问题。  此外，在调试与测试的过程中，我还进一步加深了对C++面向对象编程中“三法则”（拷贝构造函数、赋值运算符、析构函数）的理解。这次实验不仅提升了我的C++编程能力，也让我体会到设计健壮类结构的重要性，为后续处理更复杂的数据结构奠定了良好基础。 | | | | | |
| 1. **教师评语** | | | | | |
| 1. **实验成绩**   教师签名： 蒋振刚 批阅日期： 2025 年 4 月 7日 | | | | | |

注：项目性质为 演示型、验证型、设计型、综合型和创新型。