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** | 240511B13 |
| **实验项目** | 实验二 | | | **姓 名** | 徐成希 |
| **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** | 2024年3月31日 |
| 1. **实验目的和要求**   目的：练习类的构造和析构，了解他们在对象生命周期中的作用。  要求：设计和实现一个线性表类DList，满足：  1）用链表实现存储；  2）实现线性表的基本操作；  3）测试代码验证 List的基本操作。 | | | | | |
| 1. **实验环境**   系统：fedora 41  环境：g++ | | | | | |
| 1. **实验内容与过程**   **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**  **Main.cpp**  **#include <iostream>**  **#include "include/Dlist.h"**  **using namespace std;**  **int main(){**  **DList 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();**    **// 测试获取元素**  **std::cout << "\n===== 测试获取元素 =====" << std::endl;**  **if (list.getElem(2, e)) {**  **std::cout << "第2个元素是：" << e << std::endl;**  **}**    **// 测试查找元素**  **std::cout << "\n===== 测试查找元素 =====" << std::endl;**  **int pos = list.locate(20);**  **if (pos) {**  **std::cout << "元素20的位置是：" << pos << std::endl;**  **} else {**  **std::cout << "未找到元素20" << std::endl;**  **}**    **// 测试删除操作**  **std::cout << "\n===== 测试删除操作 =====" << std::endl;**  **if (list.remove(2, e)) {**  **std::cout << "删除的元素是：" << e << std::endl;**  **}**  **list.display();**    **// 测试其他操作**  **std::cout << "\n===== 测试其他操作 =====" << std::endl;**  **std::cout << "线性表长度：" << list.size() << std::endl;**  **std::cout << "线性表是否为空：" << (list.isEmpty() ? "是" : "否") << std::endl;**    **// 测试清空操作**  **std::cout << "\n===== 测试清空操作 =====" << std::endl;**  **list.clear();**  **std::cout << "清空后，线性表长度：" << list.size() << std::endl;**  **std::cout << "清空后，线性表是否为空：" << (list.isEmpty() ? "是" : "否") << std::endl;**  **list.display();**    **return 0;**  **}**  wps | | | | | |
| 1. **实验结果与分析**   实现方式  该代码采用双向链表实现线性表的功能。通过定义包含前驱指针prev和后继指针next的节点结构体Node来构建链表。  主要特点  使用头尾哨兵节点设计：  使用head和tail两个哨兵节点  简化了边界情况的处理  提高了操作的稳定性  遍历机制：  从head->next开始遍历  通过next指针依次访问节点  直到到达tail节点为止  双向链表结构使得可以双向遍历 | | | | | |
| 1. **实验心得**   1. 类设计与实现方面  理解了构造函数和析构函数在对象生命周期中的关键作用  构造函数需要正确初始化头尾节点并建立链接  析构函数要彻底释放内存避免泄漏  2. 双向链表结构特点  支持双向遍历更灵活  可直接访问前驱节点,简化了插入删除操作  需要额外存储prev指针占用更多内存  3. 关键实现技巧  使用虚拟头尾节点简化边界处理  需要仔细处理指针链接关系  注意维护长度计数的准确性  4. 代码的健壮性保证  参数有效性检查很重要  容量限制的判断  异常情况的错误提示  5. 测试工作的重要性  基本功能的正确性验证  边界条件的测试  异常处理的验证  这次实验加深了对C++面向对象编程的理解,也体会到了代码质量保证的重要性。通过实践掌握了链表这种基础数据结构的实现要点。 | | | | | |
| 1. **教师评语** | | | | | |
| 1. **实验成绩**   教师签名： 蒋振刚 批阅日期： 2024 年 3 月 31日 | | | | | |

注：项目性质为 演示型、验证型、设计型、综合型和创新型。