《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | 23软工 | 姓名 | 王天予 | 学号 | 2362401031 |
| 实验布置日期 | | 2024.9.24 | | 提交  日期 |  | | 成绩 |  |

课程实践实验2：线性表的实现和应用

## 一、问题描述及要求

### 1. 问题描述和要求

本实验的目标是实现以下三个数据结构和应用：

1. 线性表的顺序存储实现：
   * 实现无序顺序表类，提供基本的线性表操作。
   * 定义有序顺序表类，继承自无序顺序表类，用于存放递增有序的元素，增加按值插入、按位置插入和替换元素的方法。
   * 设计测试程序，测试所设计的两个类的所有方法，确保其正确性。
2. 线性表的链式存储实现：
   * 定义并实现单链表类 SingleLink。
   * 定义并实现单向循环链表类 IterSingleLink。
   * 定义并实现双向链表（非循环）类 DoubleLink。
   * 所有三个类都需要实现链表的常用操作，包括初始化、销毁、创建、插入、删除和查找。
3. 扑克牌整理应用程序：
   * 对多副不完整的扑克牌进行整理，计算可以拼凑成的完整扑克牌套数，并输出剩余的牌。
   * 采用单链表类 SingleLink 来存储扑克牌信息。

## 二、概要设计

### 1. 系统功能列表

线性表的顺序存储实现：

* 无序顺序表类：
  + 初始化顺序表。
  + 在指定位置插入元素。
  + 删除指定位置的元素。
  + 获取指定位置的元素值。
  + 修改指定位置的元素值。
  + 清空顺序表。
  + 显示顺序表的所有元素。
* 有序顺序表类（继承自无序顺序表类）：
  + 按值插入元素（保持有序）。
  + 按位置插入元素。
  + 替换指定位置的元素（保持有序）。
  + 其他继承自无序顺序表的操作。
* 测试程序：
  + 提供命令行交互界面，供用户选择执行各种操作。
  + 循环接受用户指令，直到用户选择退出。

线性表的链式存储实现：

* 抽象基类 Link：
  + 定义纯虚函数，包含链表的常用操作：初始化、销毁、创建、插入、删除、查找（get(i) 和 search(x)）等。
* 派生类：
  + SingleLink：实现单链表。
  + IterSingleLink：实现单向循环链表。
  + DoubleLink：实现双向链表（非循环）。
* 实现的操作：
  + 初始化：创建头节点。
  + 销毁：释放所有节点。
  + 创建：根据用户输入创建链表。
  + 插入：在指定位置插入元素。
  + 删除：删除指定位置的元素。
  + 查找：
    - get(i)：获取第 i 个位置的元素。
    - search(x)：查找值为 x 的元素位置。
  + 显示：输出链表中的所有元素。
* 测试程序：
  + 提供命令行界面，与用户交互，执行各种操作。
  + 支持多次操作，直到用户选择退出。

扑克牌整理应用程序：

* 文件读取模块：读取并解析输入文件，获取每副牌的牌点信息。
* 牌点存储模块：使用 SingleLink 类存储每副牌的各花色牌点。
* 完整套数计算模块：
  + 统计每张牌在所有牌中的总数量。
  + 计算可以拼凑的完整扑克牌套数。
* 剩余牌更新模块：从每副牌中移除用于拼凑完整套数的牌，更新剩余的牌点。
* 输出模块：按照要求输出拼凑的套数和每副牌剩余的牌点。

### 2. 程序运行界面设计

2.1. 线性表的顺序存储实现：

提供简单明了的命令行菜单，用户可以通过输入对应的数字或字符选择操作，例如：

请选择操作：  
1. 在无序表中插入元素  
2. 在有序表中按值插入元素  
3. 在有序表中按位置插入元素  
4. 替换有序表中的元素  
5. 显示无序表  
6. 显示有序表  
Q. 退出  
请输入您的选择：

2.2. 线性表的链式存储实现：

提供命令行交互式界面，用户可以输入指令执行各种操作，如创建链表、插入、删除、查找、获取元素、显示链表和退出程序。

2.3. 扑克牌整理应用程序：

程序读取输入文件后，直接输出所能拼凑的扑克牌套数和每副牌剩余的牌，按照指定的格式显示。

### 3. 总体设计思路

3.1. 线性表的顺序存储实现：

* 数据存储：使用动态数组（指针）来存储顺序表中的元素，并支持动态扩容。
* 类的设计：
  + SeqList 类：实现无序顺序表的基本功能。
  + OrderedSeqList 类：继承自 SeqList 类，重写和新增方法，以确保元素的有序性。
* 测试程序：在主函数中，提供循环的菜单，用户可以根据提示输入选择，调用相应的方法。

3.2. 线性表的链式存储实现：

* 类层次结构：
  + 抽象基类 Link：包含纯虚函数，定义链表的公共接口。
  + 派生类：各自实现具体的链表类型。
* 数据结构：
  + 节点结构：
    - 单链表节点包含数据和指向下一个节点的指针。
    - 单向循环链表节点与单链表相同，但尾节点指向头节点。
    - 双向链表节点包含数据、前驱指针和后继指针。
* 用户交互：
  + 程序提示用户选择要操作的链表类型。
  + 提供各种操作供用户选择执行。

3.3. 扑克牌整理应用程序：

* 数据结构：
  + 使用 SingleLink 类的实例来存储每副牌的每个花色的牌点。
  + 使用二维数组 cardCounts[4][13] 来统计每张牌的总数量。
* 算法流程：
  + 统计牌点数量：遍历所有链表，统计每张牌的总数量。
  + 计算套数：对于每张牌，计算其总数量，取所有牌的数量最小值，即为可以拼凑的完整套数 m。
  + 更新链表：根据完整套数，从每副牌的链表中删除相应数量的牌点。
  + 排序输出：在输出剩余牌点时，对链表中的牌点进行排序。

### 4. 程序结构设计

* 主程序文件：包含类的定义和实现，以及主函数 main。
* 类设计：
  + SeqList 类和 OrderedSeqList 类，用于顺序表的实现。
  + 抽象基类 Link 和派生类 SingleLink、IterSingleLink、DoubleLink，用于链式存储实现。
  + SingleLink 类也用于扑克牌整理程序的牌点存储。

## 三、详细设计

### 1. 顺序表的实现

#### 主函数设计

* 提供循环菜单，用户可以多次选择操作。
* 根据用户输入，调用 SeqList 或 OrderedSeqList 类的相应方法。
* 对于非法输入，给予提示并要求重新输入。

#### 关键算法设计

无序顺序表的操作

* 插入元素：
  + 检查插入位置的合法性（1 ≤ pos ≤ length + 1）。
  + 如果数组已满，调用扩容函数增加容量。
  + 从插入位置开始，后移元素。
  + 在指定位置插入新元素。
* 删除元素：
  + 检查删除位置的合法性（1 ≤ pos ≤ length）。
  + 从删除位置开始，前移元素，覆盖被删除的元素。
  + 长度减一。
* 获取和设置元素：
  + 检查位置的合法性。
  + 返回或修改指定位置的元素值。

有序顺序表的操作

* 按值插入元素：
  + 查找适当的位置，使得插入后顺序表仍然有序。
  + 在找到的位置插入新元素。
* 替换元素：
  + 检查位置的合法性。
  + 删除指定位置的元素。
  + 按值插入新元素，保持有序性。

### 2. 链式存储的实现

#### 抽象基类 Link

* 定义纯虚函数，包括初始化、销毁、创建、插入、删除、查找和显示等操作。

#### SingleLink 类

* 节点结构：
* struct Node {  
   int data;  
   Node\* next;  
  };
* 主要函数实现：
  + initialize()：创建头节点。
  + destroy()：遍历链表，释放所有节点。
  + create(int arr[], int n)：根据数组创建链表。
  + insert(int i, int x)：在第 i 个位置插入元素 x。
  + remove(int i)：删除第 i 个位置的元素。
  + get(int i)：获取第 i 个位置的元素值。
  + search(int x)：查找值为 x 的元素位置。
  + display()：输出链表中的所有元素。

#### IterSingleLink 类

* 节点结构：与 SingleLink 相同，但尾节点指向头节点，形成循环。
* 主要函数实现：与 SingleLink 类似，但需要处理循环链表的特性。

#### DoubleLink 类

* 节点结构：
* struct DNode {  
   int data;  
   DNode\* prev;  
   DNode\* next;  
  };
* 主要函数实现：类似于 SingleLink，但需要处理前驱指针。

### 3. 扑克牌整理应用

#### 关键算法描述

读取输入文件

* 打开输入文件，读取第一行获取 n。
* 为每副牌的每个花色创建一个 SingleLink 链表，存储牌点。
* 逐行读取文件，每 4 行读取一副牌的花色信息，按照顺序存储到链表中。

计算完整套数

* 初始化二维数组 cardCounts[4][13]，用于统计每张牌的总数量。
* 遍历所有副牌的链表，统计每张牌的总数量。
* 对于每张牌，取其数量的最小值，即为可以拼凑的完整套数 m。

更新剩余牌

* 对于每副牌的每个花色，按照完整套数 m，从链表中删除相应数量的牌点。

输出结果

* 输出完整套数 m。
* 对于每副牌，按花色顺序输出剩余的牌点：
  + 对链表进行排序。
  + 输出花色名称和排序后的牌点，牌点之间以空格分隔。

## 四、实验结果

### 1. 顺序表测试结果

测试用例 1

* 操作：
  + 在无序表位置 1 插入元素 10。
  + 在无序表位置 2 插入元素 20。
  + 在无序表位置 2 插入元素 15。
  + 显示无序表。
* 预期结果：
  + 顺序表内容为 [10, 15, 20]。
* 实际结果：
  + 程序输出了顺序表内容 [10, 15, 20]。

测试用例 2

* 操作：
  + 在有序表中按值插入元素 20。
  + 在有序表中按值插入元素 10。
  + 在有序表中按值插入元素 15。
  + 显示有序表。
* 预期结果：
  + 有序表内容为 [10, 15, 20]。
* 实际结果：
  + 程序输出了有序表内容 [10, 15, 20]。

### 2. 链表测试结果

测试用例 1：单链表

* 操作：
  1. 创建包含 [10, 20, 30] 的链表。
  2. 在位置 2 插入元素 15。
  3. 显示链表。
  4. 获取位置 3 的元素。
  5. 查找元素 20 的位置。
  6. 删除位置 2 的元素。
  7. 显示链表。
* 实际结果：
  + 插入后链表为 [10, 15, 20, 30]。
  + 获取位置 3 的元素为 20。
  + 查找元素 20 的位置为 3。
  + 删除后链表为 [10, 20, 30]。

测试用例 2：单向循环链表

* 操作：
  1. 创建包含 [1, 2, 3, 4] 的链表。
  2. 显示链表。
  3. 在位置 5 插入元素 5。
  4. 显示链表。
  5. 删除位置 1 的元素。
  6. 显示链表。
* 实际结果：
  + 插入后链表为 [1, 2, 3, 4, 5]。
  + 删除后链表为 [2, 3, 4, 5]。

### 3. 扑克牌整理测试结果

测试用例

* 输入文件内容：
* 2  
  2 1 13  
  3 2 3 4  
  1 5  
  2 6 7  
  3 8 9 10  
  2 11 12  
  2 13 1  
  1 2
* 程序运行输出：
* 0  
  Spades: 1 13  
  Hearts: 2 3 4  
  Clubs: 5  
  Diamonds: 6 7  
  Spades: 8 9 10  
  Hearts: 11 12  
  Clubs: 1 13  
  Diamonds: 2
* 测试结果分析：
  + 由于缺少部分牌，无法拼凑完整的扑克牌套数，因此输出 0。
  + 每副牌的剩余牌按照花色和牌点大小顺序输出，且没有输出数量，符合题目要求。

## 五、实验分析与探讨

### 1. 功能正确性

* 顺序表：所有基本操作都通过了测试，说明程序正确实现了无序顺序表和有序顺序表的功能，有序顺序表的有序性始终得到保持。
* 链表：各类链表的基本操作均通过测试，功能实现正确，链表结构维护良好。
* 扑克牌整理程序：程序能够正确读取并解析输入文件，计算完整套数，输出剩余牌，输出格式符合要求。

### 2. 性能分析

* 时间复杂度：
  + 顺序表的插入和删除操作需要移动元素，时间复杂度为 O(n)。
  + 链表的插入、删除、查找操作的时间复杂度为 O(n)。
  + 扑克牌整理程序中，对每张牌进行统计和处理，时间复杂度为 O(n)，其中 n 为牌的总数量。
* 空间复杂度：
  + 顺序表由于采用动态数组，空间利用率较高。
  + 链表每个节点额外需要存储指针，双向链表需要更多的指针存储空间。

### 3. 实验过程中遇到的问题

* 数组越界错误：在实现顺序表的插入和删除操作时，需要严格检查位置的合法性，避免数组下标越界。
* 内存泄漏：在链表的实现中，需要确保在销毁函数中释放所有动态分配的内存。
* 循环链表处理：需要特别注意循环链表的结束条件，避免无限循环。
* 输出格式：在扑克牌整理程序中，需仔细阅读题目要求，确保输出格式符合规定。

## 六、小结

通过本次实验，我们成功实现了线性表的顺序存储和链式存储两种基本数据结构，巩固了对线性表、顺序表、链表等数据结构的理解。通过继承和多态，成功地扩展了顺序表和链表的功能，提高了代码的重用性和可维护性。

在扑克牌整理应用程序中，我们将数据结构的知识应用于实际问题的解决，练习了文件读取、数据处理和结果输出等编程技能。通过测试和调试，提升了代码的健壮性和可靠性。

本次实验达到了预期的目标，为后续更加复杂的数据结构和算法的学习打下了坚实的基础。