《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 23软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.9.23 | | 提交  日期 | 2024.10.12 | | 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

线性表是一种最基本、最简单的数据结构，用来描述数据元素之间单一的前驱和后继关系。现实生活中，许多问题抽象出的数据模型是线性表。

本次课程实践需要实现线性表顺序存储结构、链式存储结构的基本操作，包括顺序表、单链表、双向链表、循环链表等。

题目1 线性表的顺序表实现

（1）实现无序顺序表类；

（2）定义一个顺序表类的派生类有序顺序表类，用于存放递增有序表，并为该有序顺序表类增加按值插入元素、按位置插入元素和替换元素等方法。

（3）设计一个测试程序，测试所设计的2个类是否正确。

要求：

测试程序可以测试类中所有方法的正确性。

程序每次执行可以循环接受多次命令，直至用户按“Q”或“q”退出程序。

题目2 线性表的链式存储实现

（1）请定义和实现一个单链表类SingleLink

（2）请定义和实现单向循环链表IterSingleLink类

（3）请定义和实现双向链表(非循环)DoubleLink类

（1）,（2）和（3）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

* 初始化和销毁，初始化一般创建头节点，销毁需要把链表中所有节点释放
* 创建链表

可根据输入的线性表元素（比如接受一个int数组，或者让用户输入多个int），创建出一个单链表。

* 实现插入、删除和查找（包括get（i）和search（x），i指元素序号，从1开始，x是要寻找的数据）
* 建议使用面向对象思想，先定义一个link抽象基类，里面定义上述纯虚函数，

然后再通过继承方式，定义（1）-（3）三个子类，实现里面的纯虚函数。

题目3 扑克牌整理

原有n副扑克牌，因时间久远均已张数不全。现把它们合在一起，看是否能凑成m（m<n）副完整的扑克牌（不考虑大小王）。

（1）输入：输入数据来自文本文件，文件第一行是一个数字n（1≤n≤20），表示原有n副牌。从第二行起，每4行用于描述一副牌的不同花色（固定为黑红梅方的顺序）的现有张数（各行的第一个数）和牌号i（1≤i≤13）（各行的其余数字，无序）；

（2）输出：输出的第一行是所能拼凑的扑克牌套数。接着按花色输出第j（1≤j≤n）副扑克牌中剩下的扑克，且对每个花色按牌点大小顺序输出；

（3）采用题目2中实现的带头结点的单链表类SingleLink。

## 二、概要设计

### 题目1线性表的顺序表实现

#### 一、对实验内容的理解

该实验旨在实现一个顺序表的抽象数据结构，其中包括无序顺序表和有序顺序表的实现。通过这个实验，能够加深对线性表的理解，掌握如何通过类和继承来设计数据结构。无序顺序表允许用户按任意顺序插入和操作元素，而有序顺序表则在插入元素时保持元素的递增顺序。

#### 二、系统功能列表

1. **无序顺序表**
   * 初始化顺序表
   * 添加元素（append）
   * 按索引插入元素（insert）
   * 替换指定索引的元素（replace）
   * 获取指定索引的元素（get）
   * 删除指定索引的元素（delete）
   * 显示所有元素（display）
   * 检查顺序表是否为空（isEmpty）
   * 检查顺序表是否满（isFull）
2. **有序顺序表**
   * 初始化有序顺序表
   * 按值插入元素（insertByValue）
   * 按索引插入元素（insertByLocation）
   * 替换指定索引的元素（replaceAndSort）
   * 显示所有元素（display）
3. **测试程序**
   * 提供用户交互界面，允许用户执行上述操作
   * 处理用户输入，支持多次操作，直到用户选择退出

#### 三、程序运行的界面设计

欢迎使用顺序表程序

请选择操作：

1. 创建无序顺序表

2. 创建有序顺序表

3. 对无序顺序表执行操作

4. 对有序顺序表执行操作

5. 退出程序

请输入选项：

在用户选择某个操作后，程序将提示用户输入相关数据，例如元素值、索引等。

#### 四、总体设计思路

* **数据结构**：采用数组来实现顺序表，使用动态数组来支持可变大小。无序顺序表和有序顺序表可以通过类的继承来实现。
* **设计的类**：
  + UnOrderedArrayList：无序顺序表类，提供无序插入和删除等功能。
  + OrderedArrayList：继承自UnOrderedArrayList，实现有序插入、替换和排序功能。
* **类方法**：
  + UnOrderedArrayList：
    - append(int val)：添加元素到表尾。
    - insert(int index, int val)：在指定索引插入元素。
    - replace(int index, int val)：替换指定索引的元素。
    - get(int index)：获取指定索引的元素。
    - display()：显示当前所有元素。
  + OrderedArrayList：
    - insertByValue(int val)：按值插入元素，保持有序。
    - insertByLocation(int index, int val)：按位置插入元素，插入后排序。
    - replaceAndSort(int index, int val)：替换元素后保持有序。

#### 五、程序结构设计

* **已有程序的使用**：可能使用一些标准库函数，如<iostream>用于输入输出，<algorithm>用于排序等。
* **程序文件设计**：
  + ArrayList.h：声明无序顺序表和有序顺序表类。
  + ArrayList.cpp：实现无序顺序表和有序顺序表的功能。
  + main.cpp：测试程序，包含用户交互逻辑，调用顺序表的各类方法。

#### 总体结构图

- ArrayList.h

- UnOrderedArrayList

- OrderedArrayList

- ArrayList.cpp

- UnOrderedArrayList methods

- OrderedArrayList methods

- main.cpp

- User interaction

通过这样的设计，程序可以清晰地实现无序顺序表和有序顺序表的功能，并允许用户通过命令行进行交互测试。

### 题目2：线性表的链式存储实现

#### 1. 对实验内容的理解

本实验旨在实现线性表的链式存储，通过定义和实现不同类型的链表类（单链表、单向循环链表和双向链表），掌握链式存储结构的基本操作。链表的常用操作包括初始化、销毁、创建、插入、删除和查找等。这些操作可以增强对数据结构及其相关算法的理解和应用能力。

#### 2. 系统功能列表

以下是系统的主要功能模块：

1. **链表初始化**：
   * 创建头节点。
   * 链表长度初始化。
2. **链表销毁**：
   * 遍历链表释放所有节点的内存。
3. **链表创建**：
   * 从用户输入或数组中创建链表。
4. **链表操作**：
   * 插入元素（按位置和按值）。
   * 删除元素（按位置和按值）。
   * 查找元素（按索引和按值）。
5. **链表展示**：
   * 输出链表的所有元素。

#### 3. 程序运行的界面设计

程序运行时的界面设计如下：

欢迎使用链表操作系统

请选择操作：

1. 创建链表

2. 插入元素

3. 删除元素

4. 查找元素

5. 输出链表

Q. 退出程序

请输入您的选择：

用户可以输入相应的数字或字母进行操作。

#### 4. 总体设计思路

* **数据结构**：采用链式存储结构实现线性表。
* **设计类**：
  + **抽象基类 LinkList**：
    - 纯虚函数： insert(), deleteByIndex(), deleteByNum()
    - 基类函数：getIndex(),searchNum(),display()

包含节点结构 ListNode。

* + **单链表类 SingleLink**：
    - 继承自 Link。
    - 实现所有纯虚函数。
  + **单向循环链表类 IterSingleLink**：
    - 继承自 Link。
    - 实现所有纯虚函数。
  + **双向链表类 DoubleLink**：
    - 继承自 Link。
    - 实现所有纯虚函数。

#### 5. 程序结构设计

程序结构分为以下几个部分：

1. **头文件**：
   * LinkList.h：定义抽象基类 Link 和节点结构 ListNode，单链表类 SingleLink，单向循环链表类 IterSingleLink，双向链表类 DoubleLink。
2. **实现文件**：
   * ListLink.cpp：实现单链表类的所有成员函数，单向循环链表类的所有成员函数，双向链表类的所有成员函数。
   * main.cpp：程序入口，负责用户输入和功能调用。
3. **功能实现**：
   * 各类实现链表的基本操作。
   * main.cpp 中通过用户输入驱动链表操作。

通过这个设计，我们可以确保程序结构清晰，功能分明，同时便于后续的扩展与维护。

以下是关于实验内容的理解和设计的概述。

### 题目3：扑克牌的整理

#### 1. 对实验内容的理解

本实验主要是实现一个扑克牌的整理程序。根据输入的扑克牌信息，程序需判断是否能组合成完整的扑克牌套数，并输出剩余的扑克。系统要求使用链表的形式存储扑克牌信息，并实现相关的链表操作。

#### 2. 系统的功能列表

* **输入功能**：
  + 从文本文件读取扑克牌信息。
* **处理功能**：
  + 统计各副扑克牌的数量。
  + 判断能够拼凑出的完整扑克牌套数。
  + 输出每副牌中剩余的扑克。
* **输出功能**：
  + 显示拼凑出的完整扑克套数及剩余扑克牌的详细信息。

#### 3. 程序运行的界面设计

欢迎使用扑克牌整理程序

1. 从文件输入扑克牌信息

2. 显示整理前的扑克牌信息

3. 显示整理后的扑克牌信息

#### 4. 总体设计思路

* **数据结构**：
  + 使用二维链表数组来存储每副扑克牌每个花色的信息。
  + 使用四个二维数组来计算能凑出牌的套数
* **设计的类**：
  + **SingleLink**：单链表类，用于存储扑克牌。
  + **LinkList**：抽象基类，定义基本链表操作。
* **类方法的介绍**：
  + insert(int val, int pos)：在指定位置插入新节点。
  + deleteByIndex(int index)：根据索引删除节点。
  + deleteByNum(int value)：根据值删除节点。
  + display()：输出链表的所有节点值。
* **类之间的关系描述**：
  + SingleLink 继承自 LinkList，实现链表的基本操作。

#### 5. 程序结构设计

* **已存在程序的使用**：
  + 使用已定义的 SingleLink 类和其他链表类。
* **设计的程序文件**：
  + LinkList.h：链表类的定义，包括基本操作的声明。
  + LinkList.cpp：链表类的实现。
  + main.cpp：程序入口，负责用户交互和功能调用。
* **各部分关系描述**：
  + main.cpp 中调用 cardsCount 函数读取文件并处理数据。
  + LinkList.h 和 LinkList.cpp 中定义链表结构及相关操作。

这样设计后，程序可以高效地处理扑克牌整理的问题，并提供清晰的用户交互界面。

**可包含以下内容，可根据实际情况取舍。**

**（1）对实验内容的理解。**

**（2）给出系统的功能列表（可以用图示或清单的形式）**

**（3）程序运行的界面设计 （可以用图示等方式，如：首先出现屏幕提示，请用户选择输入配置的方式，1从键盘输入活单元坐标2……3….然后用户）**

**（4）确定总体设计思路，采用何种数据结构，设计哪些类，各类的作用 ，类方法的介绍，类之间的关系描述**

**（5）程序结构设计，包括：对已有程序的使用，自己将设计哪些程序文件，各部分关系描述。**

**这部分最核心的工作是：描述解决问题所需用到的数据结构和算法。——给出明确地确定该结构及对应算法的理由。**

**此外，这部分可以给出面向对象相关的设计，例如实现完整功能所涉及的类及对象间的总体关联关系等。**

## 三、详细设计

### 题目1：线性表的顺序表实现

**主函数设计**

* 主函数实现用户交互，通过命令行输入对顺序表的操作。
* 用户可以选择插入、删除、查找等操作，输入“Q”退出程序。

**关键算法**

* **无序顺序表**：支持基本的插入、删除和查找操作。
* **有序顺序表**：增加按值插入和替换元素的功能。

**难点分析**

* 插入元素时需维护数组的顺序，避免覆盖已有数据。
* 查找元素需要考虑查找效率，确保在有序状态下快速找到目标。

### 题目2：线性表的链式存储实现

**主函数设计**

* 提供用户界面选择不同类型的链表进行操作。
* 用户可以进行链表的创建、插入、删除和查找。

**关键算法**

* **单链表**：实现基本的链表操作，包括初始化、销毁、插入、删除和查找。
* **单向循环链表**：扩展单链表，实现循环操作。
* **双向链表**：实现双向操作，便于从任一方向遍历。

**难点分析**

* 链表的指针操作较复杂，尤其是在插入和删除时需要确保链表的完整性。
* 需要正确处理头结点和尾结点的情况。

### 题目3：扑克牌整理

**主函数设计**

* 主函数负责读取扑克牌数据并调用处理函数。
* 调用 cardsCount 函数，计算并输出可以拼凑的完整副牌数量和剩余牌。

**关键算法**

* **输入处理**：读取扑克牌数据，记录每种花色和牌的数量。
* **计算完整副牌数量**：找到各花色最小牌的数量。
* **输出剩余牌**：更新每副牌中的剩余扑克并按花色输出。

**难点分析**

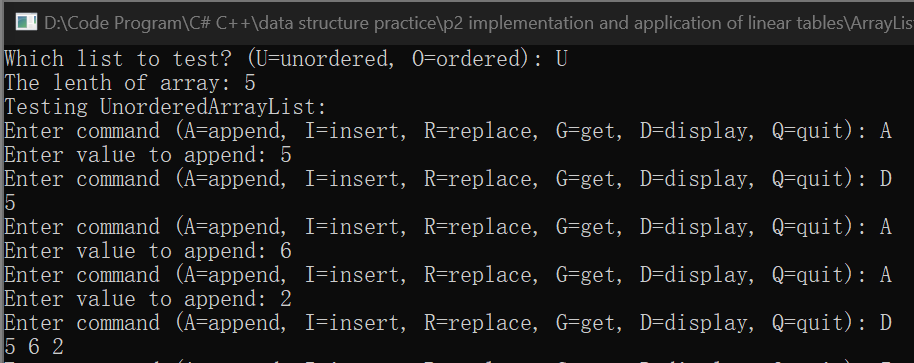
* 输入数据解析时，需要考虑牌的花色和数量的关联性。
* 在拼凑完整副牌时需要合理管理每副牌的状态，确保逻辑正确性。

整体上，题目1、2和3在实现上都是围绕线性表的数据结构进行操作，通过不同的存储方式（顺序表、链表）来实现特定的功能。每个题目在操作实现中都有独特的难点，主要集中在数据结构的操作和管理上。

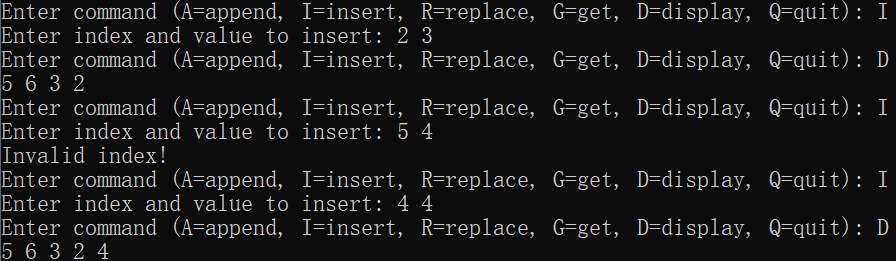
## 四、实验分析与探讨

对于题目1，先测试无序表：

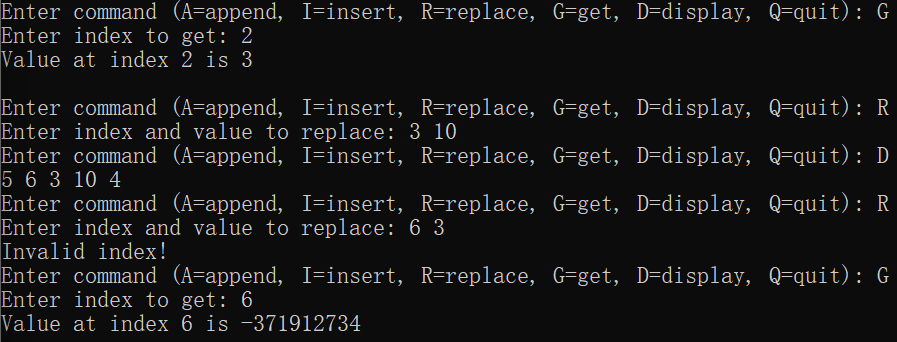
测试append函数，display函数，向无序表中追加5,6,2三个数，输出正确：



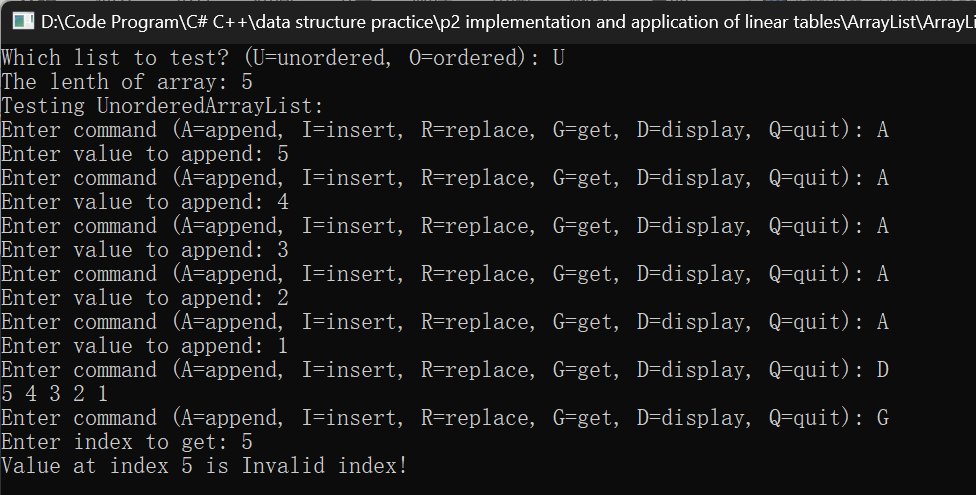
测试insert函数，向无序表中以（index,value）的方式依次插入（2,3）（5,4）（4,4），检验了插入函数的正确性和处理错误输入的灵活性，输出正确：



测试get函数和replace函数，获取下标为2的节点数据，然后将下标为3的节点数据替换成10，下标为6的节点数据替换成3，最后获取下标为6的节点数据，分别检验了两个函数的正确性和处理错误输入的灵活性，replace函数正确，而get函数对错误输入处理错误：

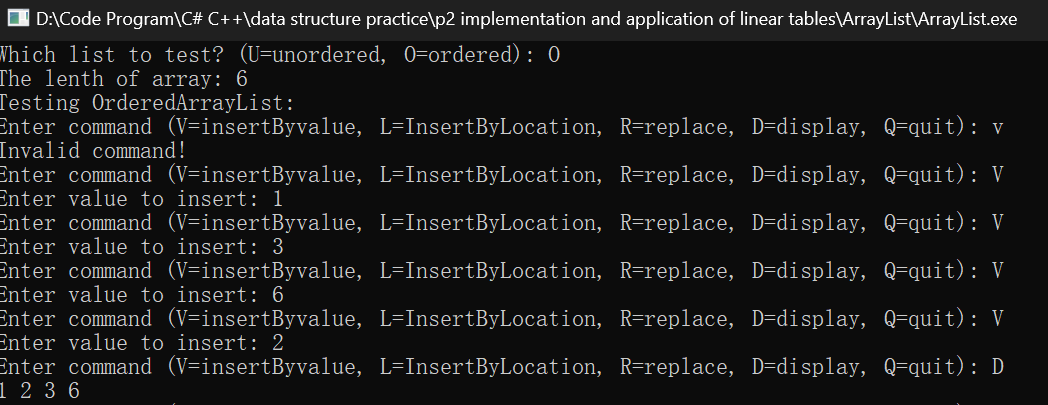


对错误处已进行修改：

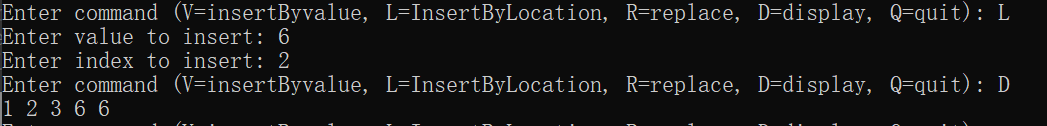


再测试有序表：

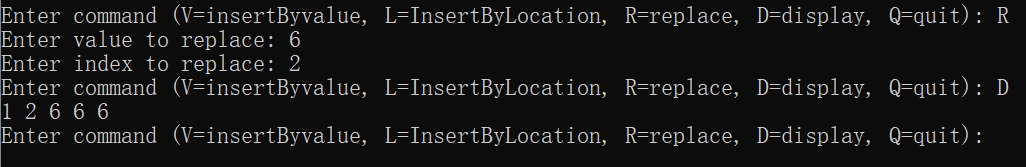
测试insertByValue函数，依次插入1,3,6,2四个数，输出正确：



测试InsertByLocation函数，其实该函数是与有序表定义相矛盾的，我只能强行进行一次重排序，或者说就是insertByNum函数而已，没有讨论的必要，输出正确：



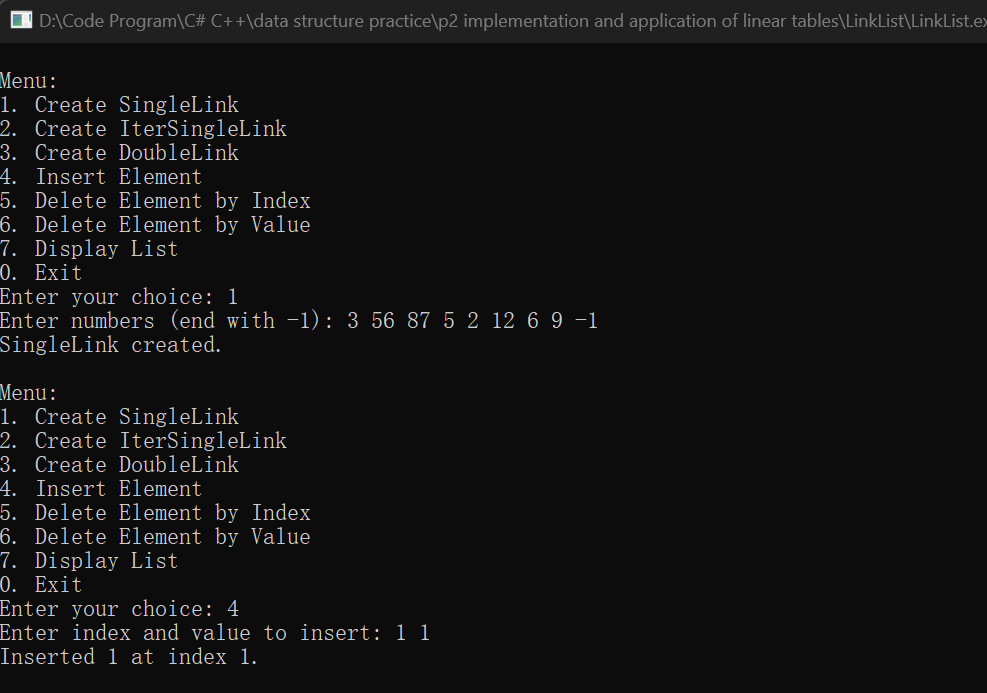
测试replace函数，完全同InsertByLocation函数，没有讨论的必要，输出正确：

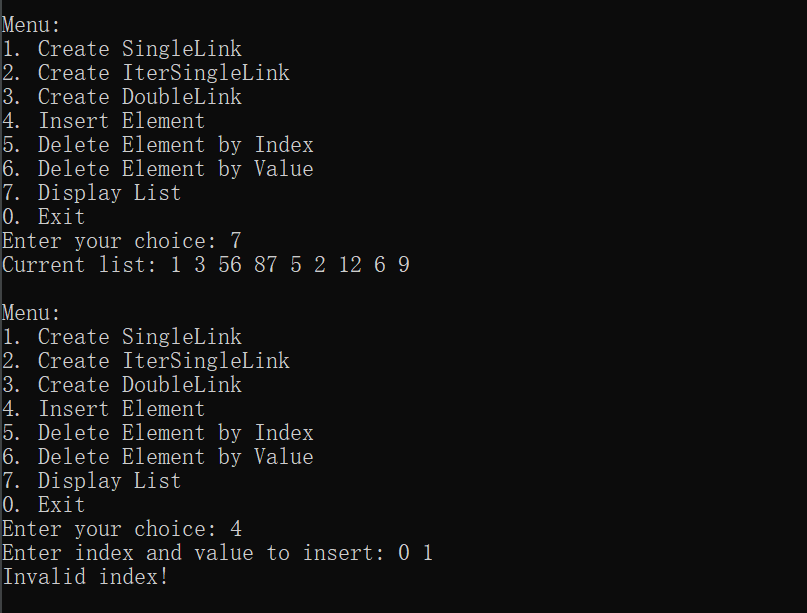


总结：题目1就是简单的实现有序表和无序表类，没有太多值得讨论的地方，需要注意的就是异常输入的处理，时间复杂度和算法也没什么可深入讨论的，基本都是一两层必须循环的事情，算是本次实验的基础练手题。

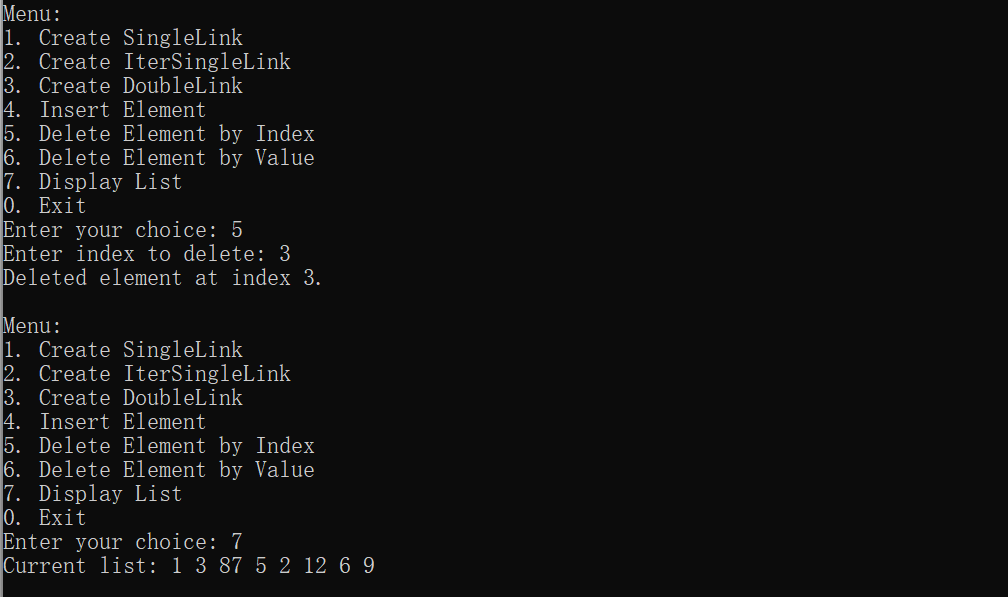
对于题目2，先测试SingleLink：

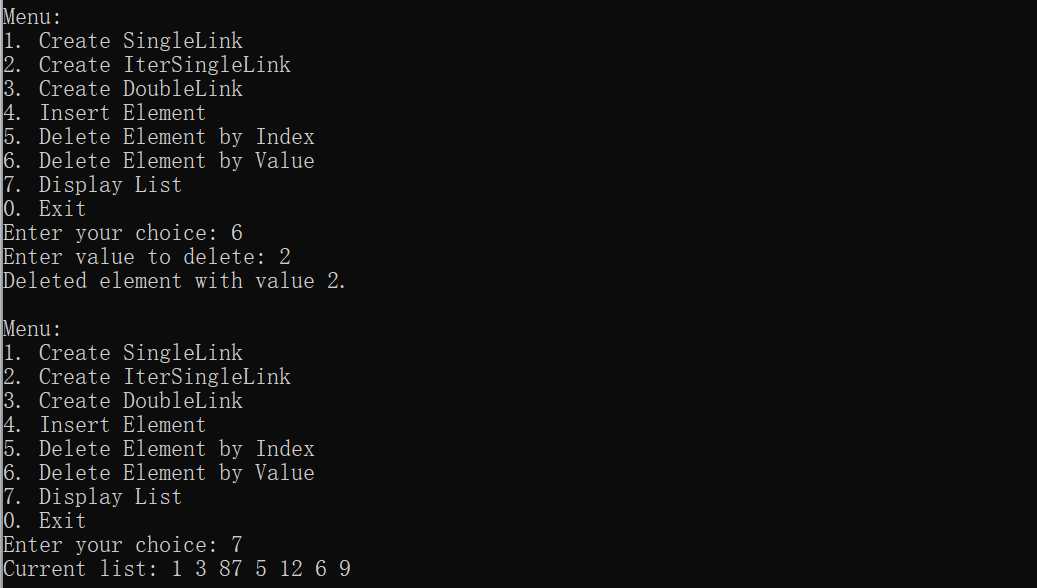
测试构造函数，Insert函数，display函数，创建3 56 87 5 2 12 6 9的单链表，然后插入（1,1）（0,1），测试函数的正确性和对错误输入的处理能力，输出正确：

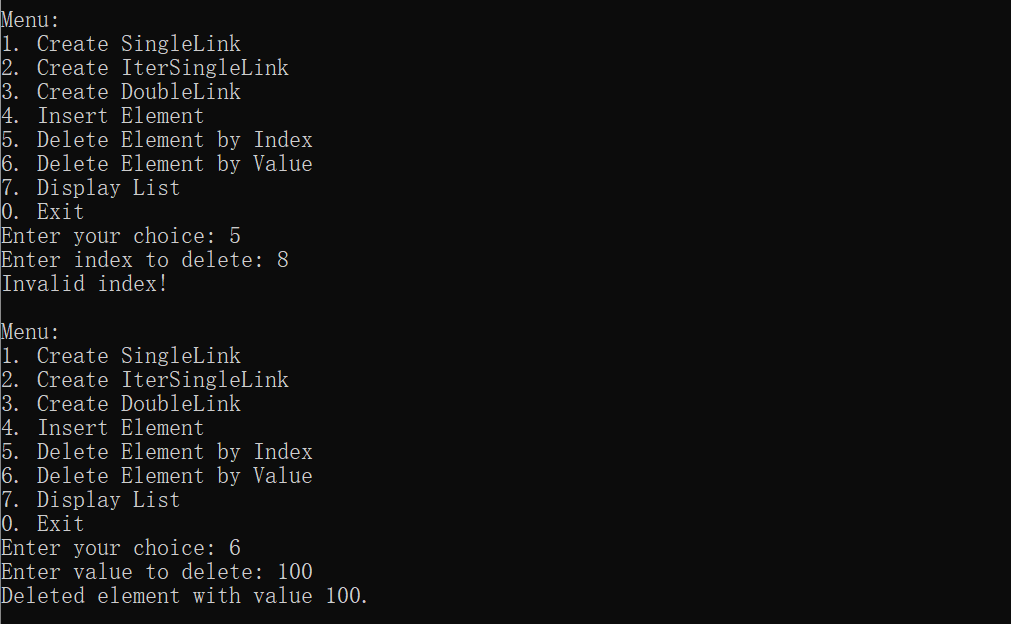




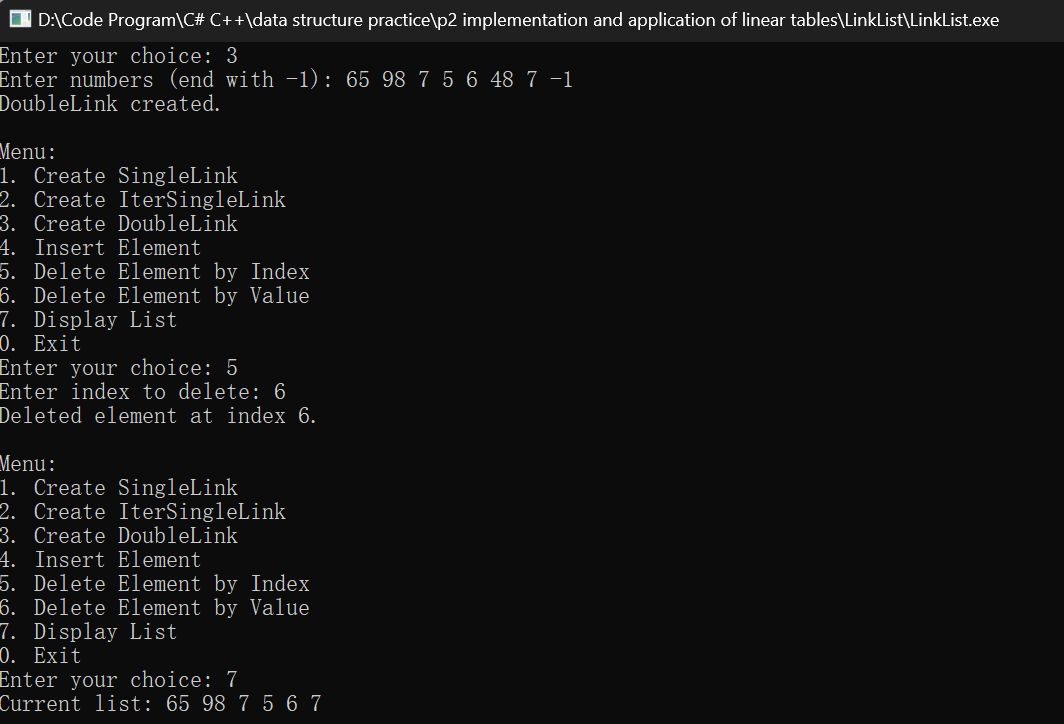
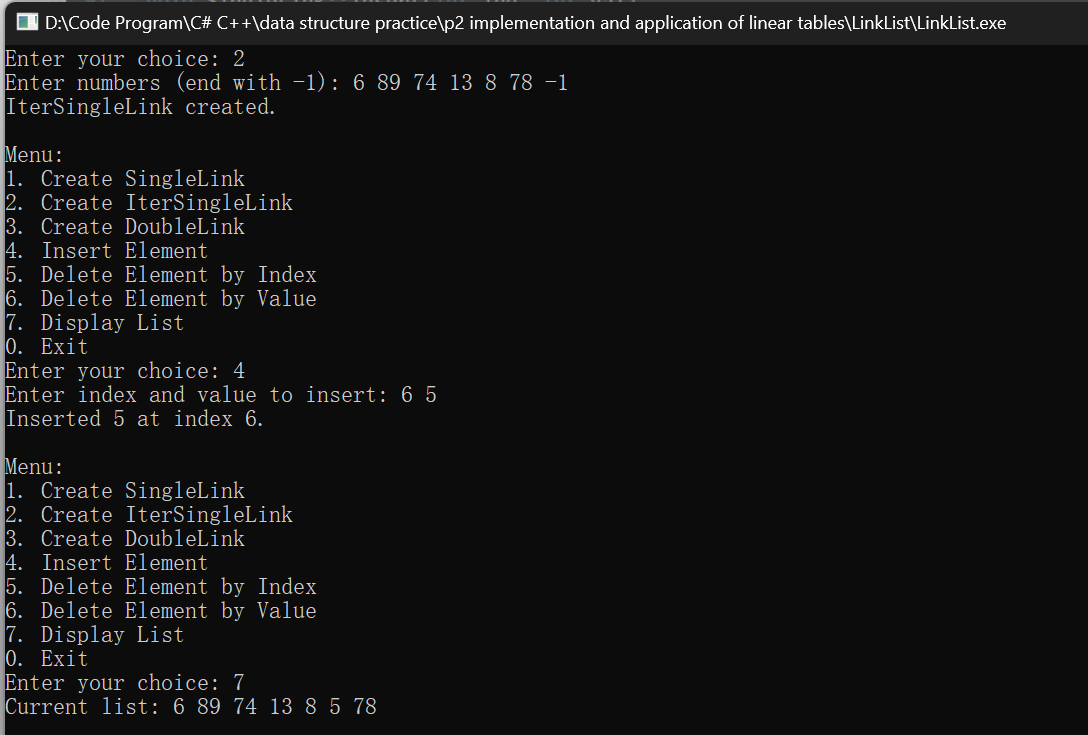
测试deleteByIndex和deleteByValue函数，先是删除下标为3的节点，删除数值为2的节点，再是删除下标为8的节点，删除数值为100的节点。虽然没有100的节点，但这里不报错也没有事情，反正就是遍历了一遍没找到，不影响程序安全性，输出正确。





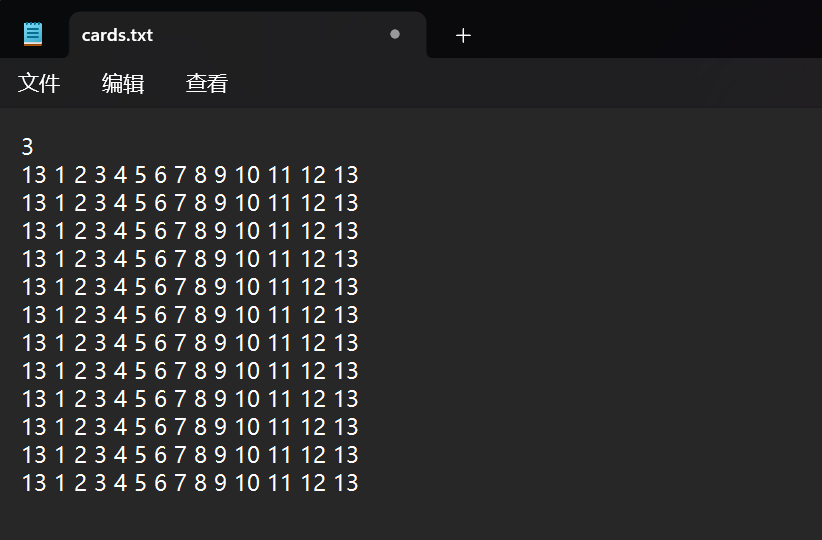


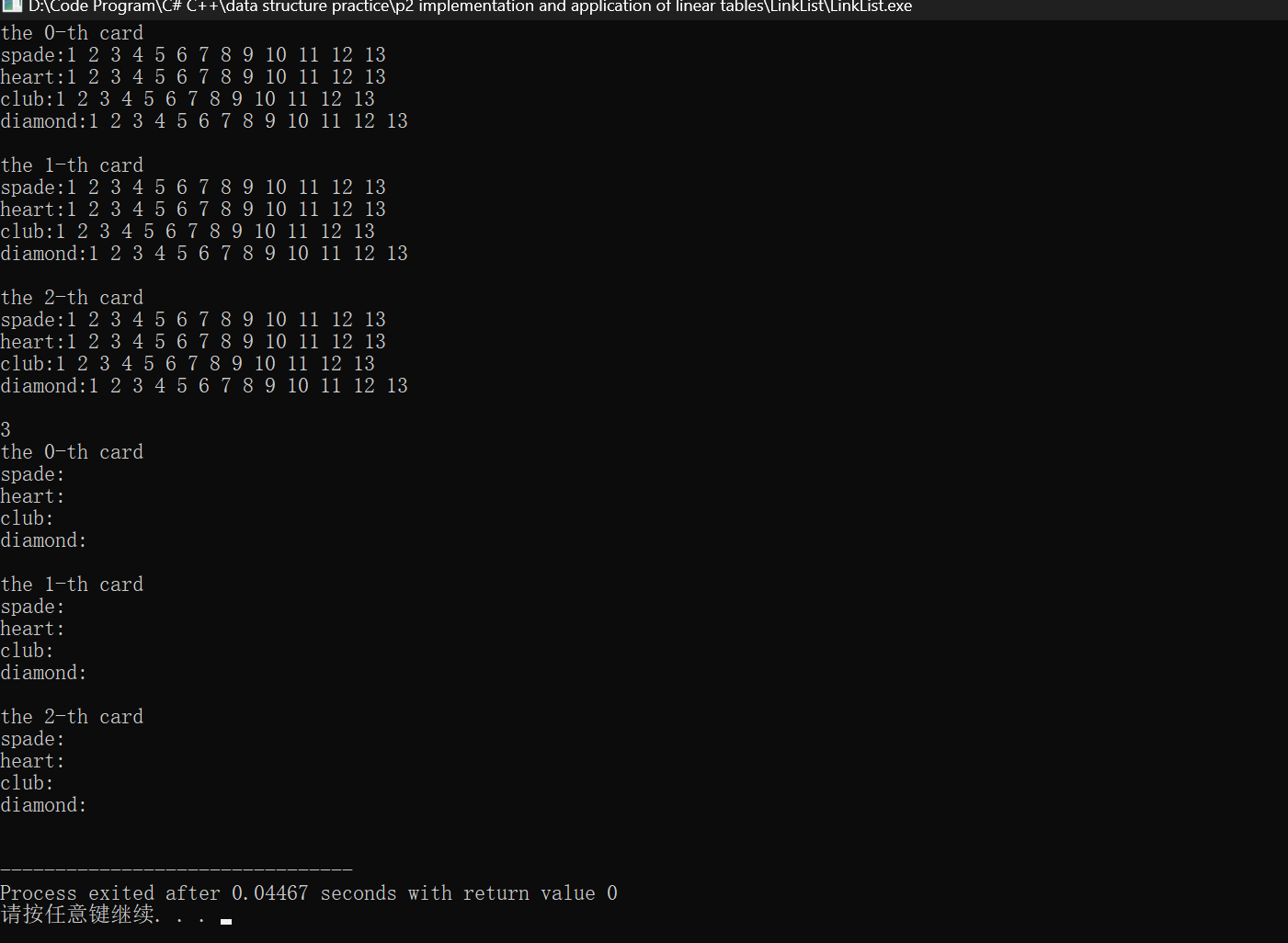
至于其余两个链表，循环和双向链表，因为本次实验并没有涉及到他们与单链表不同的地方，少许的区别就是在进行上述函数时，要对应改变一下节点的指针域，有的是会最后指向头结点，有的是还有last指针域，这里就不多赘述了，测试均同单链表且正确：



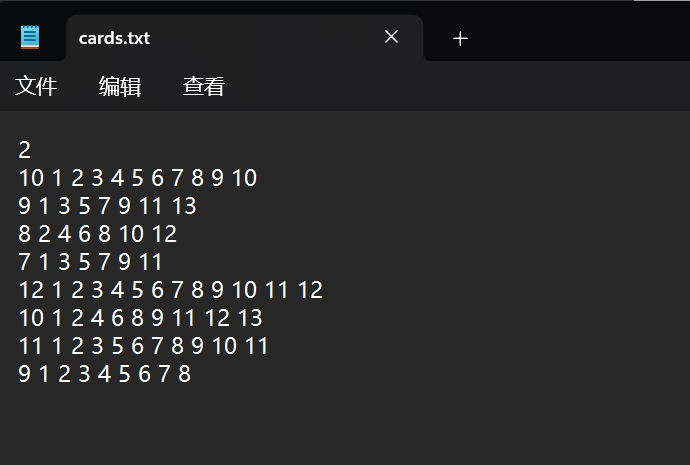
总结：本题的关键在于基类和派生类，虚拟函数和函数实现两方面的知识掌握度，比如我这次就是初次接触这两个概念，发现这种方法极大的便利了代码编写的同时，也遇到了前所未有的错误难点，具体的函数实现其实就是编写对应链表的各种方法，算法也没涉及到什么，链表最关键的就是指针域的处理，像这次我就同时用了头尾的虚拟节点，前后的指针域，为的就是程序的顺利运行，数据处理时不要越界报错。

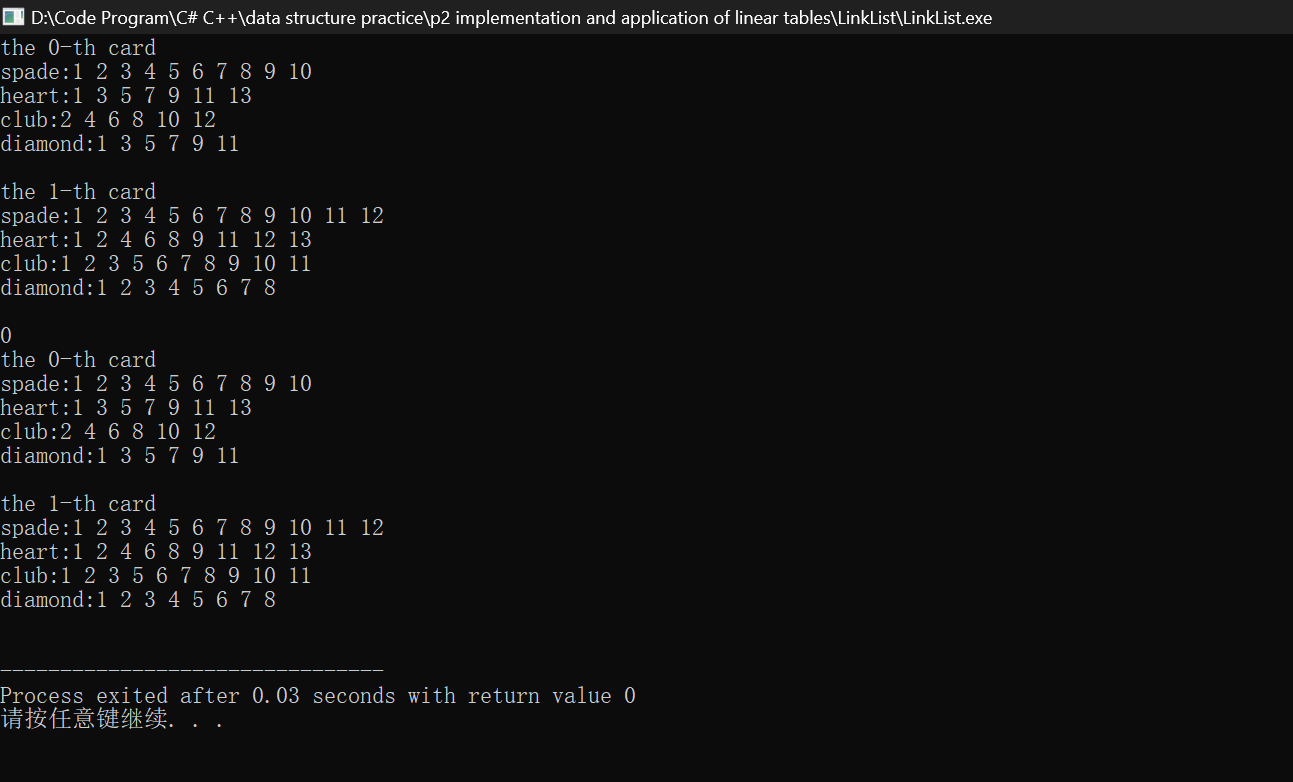
对于题目3,先测试完整的三副牌：输出正确：



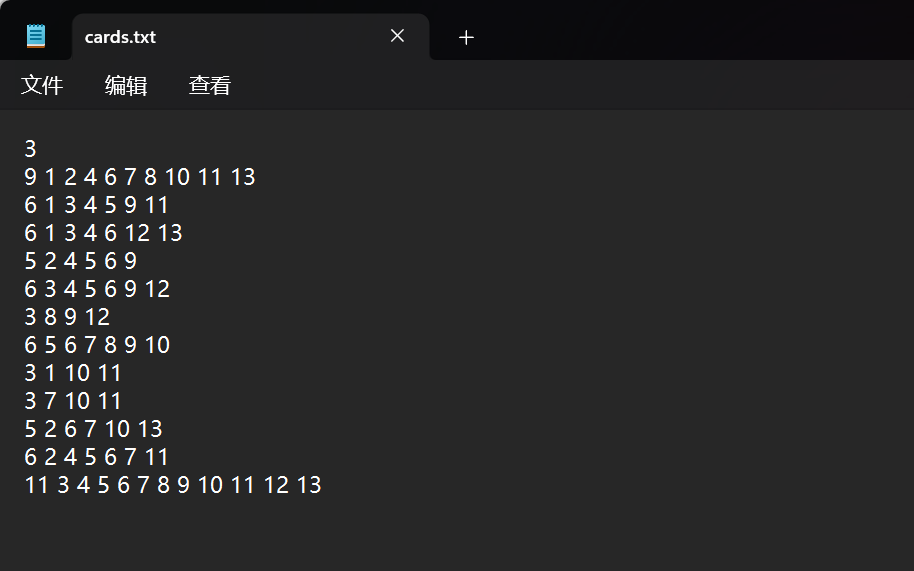


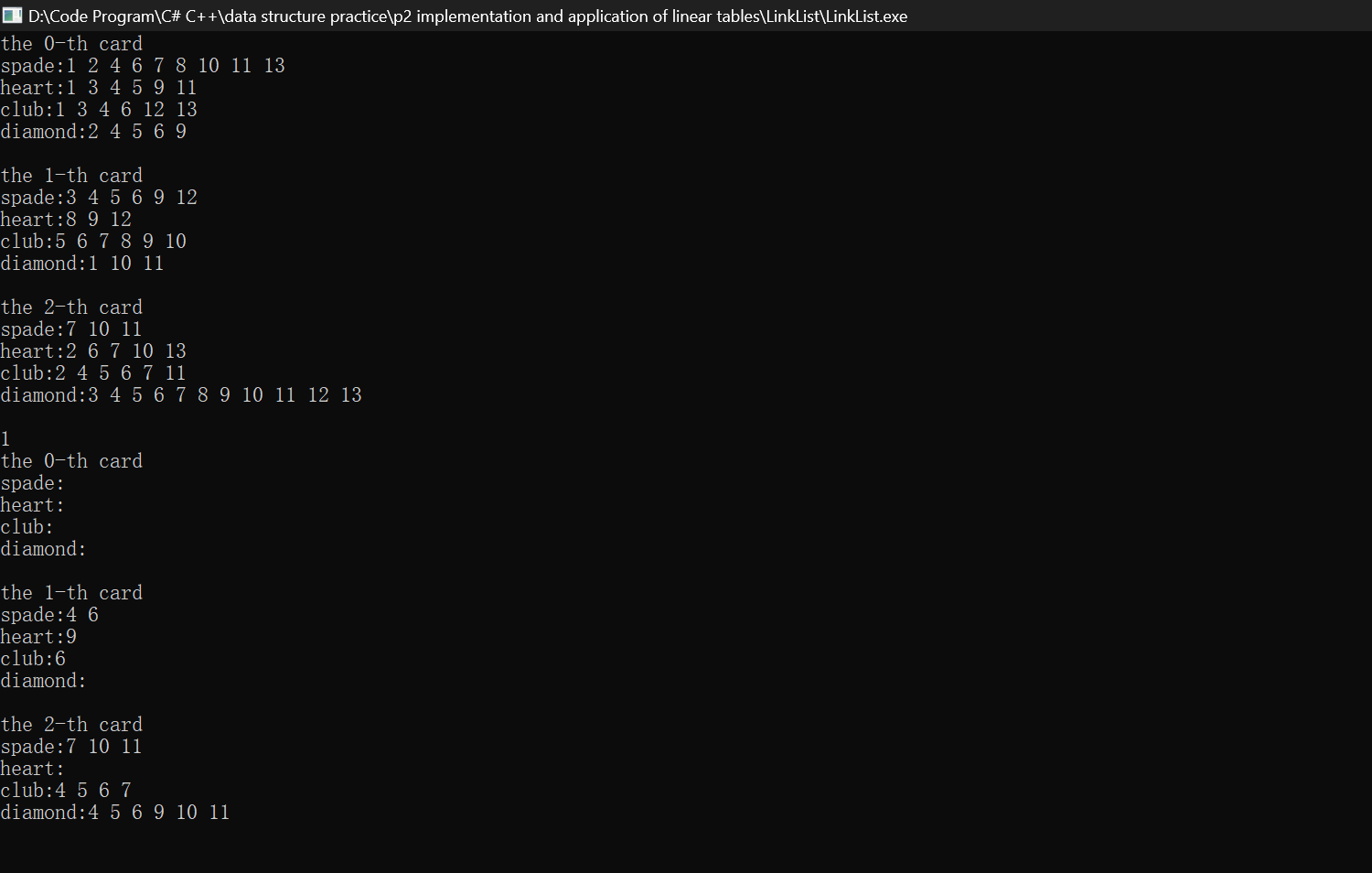
再测试无法凑出牌的情况：输出正确





最后测试能凑出牌，但是有剩牌的情况：





总结：这道题继承了上道题的SingleLink类，并且我也是在LinkList项目中的main.cpp文件中实现的对应函数，只是把第二道题的测试部分注释掉了，说回题目本身，找出能凑出多少副牌不难，只要建一个四种花色牌的哈希表就行了，然后根据凑牌数量来对每副牌的花色链表进行删减，但我也有问题想提，即该题和链表的相关性到底有多大，我实现的方法中，只是将牌组用二维链表数组存储了，似乎也没起到多关键的作用。核心的算法依旧是普通的循环遍历，是否存在更好的处理方式，此次试验后我会再去查询相关资料，毕竟我的实现也比较粗糙，很多类例如card类我都没有，只是针对性的处理，模块性还不够。

## 五、小结

在整个设计与实现过程中，三道题目涉及了顺序表和链表等基础数据结构的应用与扩展。通过完成这些题目，我对数据结构的操作、面向对象的设计思维，以及算法的实现细节有了更加深入的理解。以下是对三道题目的实现小结与未来展望：

### 1. 题目一：线性表的顺序表实现

* **完成内容**：实现了无序顺序表、有序顺序表的基本功能，包括按位置、按值插入等操作。程序通过继承与派生的方式实现了有序顺序表的功能扩展，同时也能够对表内元素进行替换等操作。
* **未完成内容**：目前对于边界条件（如极限情况下的大量数据）还未做详细的性能优化。
* **局限性**：顺序表在插入、删除操作上效率相对较低，未来可以尝试通过跳表等数据结构进行进一步优化。
* **展望**：通过优化存储结构与设计，提升顺序表在大数据下的运行效率。此外，未来可以加入异常处理机制，更加全面地处理非法输入。

### 2. 题目二：链表的链式存储实现

* **完成内容**：实现了单链表、循环单链表和双向链表的基本操作，包括插入、删除和查找功能。各链表的操作均采用面向对象的设计模式，并基于抽象类LinkList，从而简化了代码复用性。
* **未完成内容**：针对链表的性能分析及其与其他数据结构的比较还可以深入。未实现链表的复杂操作（如倒序查找、批量删除等）。
* **局限性**：链表操作的时间复杂度为O(n)，当数据规模较大时，效率并不是最优。链表的应用场景可以根据需求的不同进行更加细化的讨论和选择。
* **展望**：进一步探讨链表的优化算法，如在双向链表中引入缓存机制以提高查找和插入效率。

### 3. 题目三：扑克牌整理

* **完成内容**：实现了扑克牌整理程序，能够通过输入文本文件，对扑克牌进行统计和整理，判断能够凑出的完整扑克牌数量，并输出剩余的牌。
* **未完成内容**：目前程序较为基础，未实现更多的功能扩展，如自动从不同副牌中分配剩余牌。
* **局限性**：对于输入文件的格式有严格要求，如果输入格式不符合预期，程序会报错，缺乏异常处理。其次，扑克牌的排序与分配逻辑可以更加智能化。
* **展望**：可以加入错误输入的容错处理，并且对扑克牌进行更加复杂的统计分析操作，比如自动根据剩余牌型来推荐如何进一步整理。

### 感想与收获

在这三道题目的设计与实现过程中，收获了不少对数据结构的实际应用经验：

* 面向对象编程的思想得到了强化，尤其在继承与扩展方面的设计。
* 数据结构的基本操作得到了实际验证，掌握了如何通过简单的操作组合处理复杂的任务（如扑克牌整理）。
* 在链表的实现过程中，理解了链表的动态存储优势，也看到了链表操作中潜在的性能瓶颈。

### 完善与未来方向

* **性能优化**：对于大规模数据操作，可以尝试更高效的存储结构，如跳表、红黑树等，提升性能。
* **异常处理**：目前程序的异常处理机制不够完善，可以加入输入校验、文件格式检查等防护措施，确保程序的健壮性。
* **扩展功能**：未来可以进一步扩展题目的功能，如添加批量处理功能或图形化界面，让操作更加直观。

通过这次实验，我不仅对顺序表和链表的实现有了更深的理解，还在程序设计过程中学会了如何权衡代码的可读性与效率。希望在后续学习中，能够掌握更多的高级数据结构，并将其应用于更复杂的实际问题。

## 附录：源代码

1.实验环境：Dev-C++ 5.11集成开发环境，C++11标准。

2、

（1）ArrayList.h

//ArrayList.h

#ifndef ARRAYLIST\_H

#define ARRAYLIST\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

class UnOrderedArrayList

{

protected:

int\* data;

int capacity;

int size;

public:

UnOrderedArrayList(int capacity = 10);

~UnOrderedArrayList();

bool isEmpty()const;

bool isFull()const;

void append(int);

void insert(int, int);

void replace(int, int);

void get(int)const;

void display()const;

void getSize()const;

};

class OrderedArrayList: public UnOrderedArrayList

{

public:

OrderedArrayList(int capacity = 10);

void insertByValue(int);

void insertByLocation(int, int);

void replaceAndSort(int, int);

};

#endif

（2）ArrayList.cpp

//ArrayList.cpp

#include "ArrayList.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

UnOrderedArrayList::UnOrderedArrayList(int capacity)

{

this->capacity = capacity;

data = new int[capacity];

size = 0;

}

UnOrderedArrayList::~UnOrderedArrayList()

{

delete[] data;

}

bool UnOrderedArrayList::isEmpty()const

{

return size == 0;

}

bool UnOrderedArrayList::isFull()const

{

return size == capacity;

}

void UnOrderedArrayList::append(int val)

{

if(isFull())

{

cout << "ArrayList is Full!" << endl;

return;

}

data[size ++] = val;

}

void UnOrderedArrayList::insert(int ind, int val)

{

if(ind < 0 || ind > size)

{

cout << "Invalid index!" << endl;

}

else if(isFull())

{

cout << "ArrayList is Full!" << endl;

}

else

{

for(int i = size + 1; i > ind; i --)

{

data[i] = data[i - 1];

}

data[ind] = val;

size ++;

}

}

void UnOrderedArrayList::replace(int ind, int val)

{

if(ind < 0 || ind > size)

{

cout << "Invalid index!" << endl;

return;

}

data[ind] = val;

}

void UnOrderedArrayList::get(int ind)const

{

if(ind < 0 || ind >= size)

{

cout << "Invalid index!" << endl;

return;

}

cout << data[ind] << endl;

return;

}

void UnOrderedArrayList::display()const

{

for(int i = 0; i < size; i ++)

cout << data[i] << " ";

cout << endl;

return;

}

void UnOrderedArrayList::getSize()const

{

cout << size << endl;

return;

}

OrderedArrayList::OrderedArrayList(int capacity) : UnOrderedArrayList(capacity) {}

void OrderedArrayList::insertByValue(int val)

{

if(isFull())

{

cout << "ArrayList is Full!" << endl;

return;

}

int i = size - 1;

while(i >= 0 && data[i] > val)

{

data[i + 1] = data[i];

i --;

}

data[i + 1] = val;

size ++;

}

void OrderedArrayList::insertByLocation(int ind, int val)

{

insert(ind, val);

sort(data, data + size);

}

void OrderedArrayList::replaceAndSort(int ind, int val)

{

replace(ind, val);

sort(data, data + size);

}

（3）main.cpp（题目1）

//main.cpp

#include "ArrayList.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void testUnOrderedArrayList(int count)

{

UnOrderedArrayList list(count);

char command;

int value, index;

cout << "Testing UnorderedArrayList:" << endl;

while (true)

{

cout << "Enter command (A=append, I=insert, R=replace, G=get, D=display, Q=quit): ";

cin >> command;

if (command == 'Q' || command == 'q') break;

switch (command)

{

case 'A':

cout << "Enter value to append: ";

cin >> value;

list.append(value);

break;

case 'I':

cout << "Enter index and value to insert: ";

cin >> index >> value;

list.insert(index, value);

break;

case 'R':

cout << "Enter index and value to replace: ";

cin >> index >> value;

list.replace(index, value);

break;

case 'G':

cout << "Enter index to get: ";

cin >> index;

cout << "Value at index " << index << " is ";

list.get(index);

cout << endl;

break;

case 'D':

list.display();

break;

default:

cout << "Invalid command!" << endl;

}

}

}

void testOrderedArrayList(int count)

{

OrderedArrayList list(count);

char command;

int value, index;

cout << "Testing OrderedArrayList:" << endl;

while (true)

{

cout << "Enter command (V=insertByvalue, L=InsertByLocation, R=replace, D=display, Q=quit): ";

cin >> command;

if (command == 'Q' || command == 'q') break;

switch (command)

{

case 'V':

cout << "Enter value to insert: ";

cin >> value;

list.insertByValue(value);

break;

case 'L':

cout << "Enter value to insert: ";

cin >> value;

cout << "Enter index to insert: ";

cin >> index;

list.insertByLocation(index, value);

break;

case 'R':

cout << "Enter value to replace: ";

cin >> value;

cout << "Enter index to replace: ";

cin >> index;

list.replaceAndSort(index, value);

break;

case 'D':

list.display();

break;

default:

cout << "Invalid command!" << endl;

}

}

}

int main()

{

char listType;

int count = 0;

cout << "Which list to test? (U=unordered, O=ordered): ";

cin >> listType;

cout << "The lenth of array: ";

cin >> count;

if (listType == 'U' || listType == 'u')

testUnOrderedArrayList(count);

else if (listType == 'O' || listType == 'o')

testOrderedArrayList(count);

else

cout << "Invalid list type!" << endl;

return 0;

}

（4）LinkList.h

//LinkList.h

#ifndef LINKLIST\_H

#define LINKLIST\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

struct ListNode

{

int val;

ListNode\* next;

ListNode\* last;

ListNode(int x): val(x), next(nullptr), last(nullptr){}

};

class LinkList

{

public:

ListNode\* dummyHead;

ListNode\* dummyTail;

int listlen;

~LinkList();

int getIndex(int)const;

int searchNum(int)const;

void display()const;

virtual void insert(int, int) = 0;

virtual void deleteByIndex(int) = 0;

virtual bool deleteByNum(int) = 0;

};

class SingleLink : public LinkList

{

public:

SingleLink();

SingleLink(vector<int>&);

void insert(int, int) override;

void deleteByIndex(int) override;

bool deleteByNum(int) override;

};

class IterSingleLink : public LinkList

{

public:

IterSingleLink();

IterSingleLink(vector<int>&);

void insert(int, int) override;

void deleteByIndex(int) override;

bool deleteByNum(int) override;

};

class DoubleLink : public LinkList

{

public:

DoubleLink();

DoubleLink(vector<int>&);

void insert(int, int) override;

void deleteByIndex(int) override;

bool deleteByNum(int) override;

};

#endif

（5）LinkList.cpp

//LinkList.cpp

#include "LinkList.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

LinkList::~LinkList()

{

ListNode\* current = dummyHead;

ListNode\* temp;

while (current != nullptr)

{

temp = current;

current = current->next;

delete temp;

}

}

int LinkList::getIndex(int ind)const

{

ListNode\* p1 = dummyHead;

while(ind --)

p1 = p1->next;

return p1->val;

}

int LinkList::searchNum(int num)const

{

int count = 1;

ListNode\* p1 = dummyHead->next;

while(p1->val != num)

{

p1 = p1->next;

count ++;

}

return count;

}

void LinkList::display()const

{

ListNode\* p1 = dummyHead;

while(p1->next != nullptr && p1->next != dummyHead)

{

p1 = p1->next;

cout << p1->val << " ";

}

}

SingleLink::SingleLink()

{

dummyHead = new ListNode(0);

listlen = 0;

}

SingleLink::SingleLink(vector<int> &nums)

{

dummyHead = new ListNode(0);

listlen = 0;

ListNode\* p1 = dummyHead;

for(int i = 0; i < nums.size(); i ++)

{

ListNode\* temp = new ListNode(nums[i]);

p1->next = temp;

p1 = temp;

listlen ++;

}

}

void SingleLink::insert(int ind, int val)

{

if(ind < 1 || ind > listlen + 1)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* temp = new ListNode(val);

for(int i = 1; i < ind; i ++)

p1 = p1->next;

temp->next = p1->next;

p1->next = temp;

listlen ++;

cout << "Inserted " << val << " at index " << ind << "." << endl;

}

void SingleLink::deleteByIndex(int ind)

{

int index = ind;

if(ind < 1 || ind > listlen)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(ind --)

{

p2 = p1;

p1 = p1->next;

}

p2->next = p1->next;

delete p1;

listlen --;

cout << "Deleted element at index " << index << "." << endl;

}

bool SingleLink::deleteByNum(int num)

{

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(p1->next != nullptr)

{

p2 = p1;

p1 = p1->next;

if(p1->val == num)

{

p2->next = p1->next;

listlen --;

delete p1;

return true;

}

}

return false;

}

IterSingleLink::IterSingleLink()

{

dummyHead = new ListNode(0);

dummyHead->next = dummyHead;

listlen = 0;

}

IterSingleLink::IterSingleLink(vector<int> &nums)

{

dummyHead = new ListNode(0);

listlen = 0;

ListNode\* p1 = dummyHead;

for(int i = 0; i < nums.size(); i ++)

{

ListNode\* temp = new ListNode(nums[i]);

p1->next = temp;

p1 = temp;

listlen ++;

}

}

void IterSingleLink::insert(int ind, int val)

{

if(ind < 1 || ind > listlen + 1)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* temp = new ListNode(val);

for(int i = 1; i < ind; i ++)

p1 = p1->next;

temp->next = p1->next;

p1->next = temp;

listlen ++;

cout << "Inserted " << val << " at index " << ind << "." << endl;

}

void IterSingleLink::deleteByIndex(int ind)

{

int index = ind;

if(ind < 1 || ind > listlen)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(ind --)

{

p2 = p1;

p1 = p1->next;

}

p2->next = p1->next;

delete p1;

listlen --;

cout << "Deleted element at index " << index << "." << endl;

}

bool IterSingleLink::deleteByNum(int num)

{

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(p1->next != dummyHead)

{

p2 = p1;

p1 = p1->next;

if(p1->val == num)

{

p2->next = p1->next;

listlen --;

delete p1;

return true;

break;

}

}

return false;

}

DoubleLink::DoubleLink()

{

dummyHead = new ListNode(0);

dummyHead->next = dummyTail;

dummyTail->last = dummyHead;

dummyTail = new ListNode(0);

listlen = 0;

}

DoubleLink::DoubleLink(vector<int> &nums)

{

dummyHead = new ListNode(0);

listlen = 0;

ListNode\* p1 = dummyHead;

for(int i = 0; i < nums.size(); i ++)

{

ListNode\* temp = new ListNode(nums[i]);

p1->next = temp;

temp->last = p1;

p1 = temp;

listlen ++;

}

}

void DoubleLink::insert(int ind, int val)

{

if(ind < 1 || ind > listlen + 1)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* temp = new ListNode(val);

for(int i = 1; i < ind; i ++)

p1 = p1->next;

temp->next = p1->next;

p1->next->last = temp;

p1->next = temp;

temp->last = p1;

listlen ++;

cout << "Inserted " << val << " at index " << ind << "." << endl;

}

void DoubleLink::deleteByIndex(int ind)

{

int index = ind;

if(ind < 1 || ind > listlen)

{

cout << "Invalid index!" << endl;

return;

}

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(ind --)

{

p2 = p1;

p1 = p1->next;

}

p2->next = p1->next;

p1->next->last = p2;

delete p1;

listlen --;

cout << "Deleted element at index " << index << "." << endl;

}

bool DoubleLink::deleteByNum(int num)

{

ListNode\* p1 = dummyHead;

ListNode\* p2;

while(p1->next != nullptr)

{

p2 = p1;

p1 = p1->next;

if(p1->val == num)

{

p2->next = p1->next;

p1->next->last = p2;

listlen --;

delete p1;

return true;

break;

}

}

return false;

}

（6）main.cpp（题目2,3）

//main.cpp

#include "LinkList.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void displayMenu()

{

cout << "\nMenu:\n";

cout << "1. Create SingleLink\n";

cout << "2. Create IterSingleLink\n";

cout << "3. Create DoubleLink\n";

cout << "4. Insert Element\n";

cout << "5. Delete Element by Index\n";

cout << "6. Delete Element by Value\n";

cout << "7. Display List\n";

cout << "0. Exit\n";

}

void testFunction()

{

LinkList\* list = nullptr;

int choice;

while (true)

{

displayMenu();

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

{

vector<int> nums;

int num;

cout << "Enter numbers (end with -1): ";

while (true)

{

cin >> num;

if (num == -1) break;

nums.push\_back(num);

}

list = new SingleLink(nums);

cout << "SingleLink created." << endl;

break;

}

case 2:

{

vector<int> nums;

int num;

cout << "Enter numbers (end with -1): ";

while (true)

{

cin >> num;

if (num == -1) break;

nums.push\_back(num);

}

list = new IterSingleLink(nums);

cout << "IterSingleLink created." << endl;

break;

}

case 3:

{

vector<int> nums;

int num;

cout << "Enter numbers (end with -1): ";

while (true)

{

cin >> num;

if (num == -1) break;

nums.push\_back(num);

}

list = new DoubleLink(nums);

cout << "DoubleLink created." << endl;

break;

}

case 4:

{

if (!list)

{

cout << "Please create a list first!" << endl;

break;

}

int ind, val;

cout << "Enter index and value to insert: ";

cin >> ind >> val;

list->insert(ind, val);

break;

}

case 5:

{

if (!list)

{

cout << "Please create a list first!" << endl;

break;

}

int ind;

cout << "Enter index to delete: ";

cin >> ind;

list->deleteByIndex(ind);

break;

}

case 6:

{

if (!list)

{

cout << "Please create a list first!" << endl;

break;

}

int val;

cout << "Enter value to delete: ";

cin >> val;

list->deleteByNum(val);

cout << "Deleted element with value " << val << "." << endl;

break;

}

case 7:

{

if (!list)

{

cout << "Please create a list first!" << endl;

break;

}

cout << "Current list: ";

list->display();

cout << endl;

break;

}

case 0:

cout << "Exiting..." << endl;

delete list;

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

void cardsInput(stringstream &ss, vector<int> &color, vector<int> &cards)

{

int number;

while(ss >> number)

{

color[number] ++;

cards.push\_back(number);

}

sort(cards.begin(), cards.end());

}

void cardsDisplay(int n, vector<vector<SingleLink\*> > &allCards)

{

for(int i = 0; i < n; i ++)

{

cout << "the " << i << "-th card" << endl;

for(int j = 0; j < 4; j ++)

{

switch (j)

{

case 0:

cout << "spade:";

break;

case 1:

cout << "heart:";

break;

case 2:

cout << "club:";

break;

case 3:

cout << "diamond:";

break;

}

ListNode\* p = allCards[i][j]->dummyHead->next;

while(p != nullptr)

{

cout << p->val << " ";

p = p->next;

}

cout << endl;

}

cout << endl;

}

}

void cardsCount(string &filename)

{

ifstream ifs(filename);

if(!ifs)

{

cerr << "Unable to open the file for reading!" << endl;

return;

}

string line;

int n, count;

ifs >> n;

getline(ifs, line);

vector<vector<SingleLink\*> > allCards(n, vector<SingleLink\*>(4));

vector<int> spade(14, 0);

vector<int> heart(14, 0);

vector<int> club(14, 0);

vector<int> diamond(14, 0);

for(int i = 0; i < n; i ++)

{

for(int j = 0; j < 4; j ++)

{

getline(ifs, line);

vector<int> cards;

stringstream ss(line);

ss >> count;

switch (j)

{

case 0:

cardsInput(ss, spade, cards);

break;

case 1:

cardsInput(ss, heart, cards);

break;

case 2:

cardsInput(ss, club, cards);

break;

case 3:

cardsInput(ss, diamond, cards);

break;

}

allCards[i][j] = new SingleLink(cards);

}

}

int minNum = n;

vector<vector<int> > arrays = {spade, heart, club, diamond};

for(const auto& array : arrays)

{

for(int i = 1; i < 14; i ++)

{

if(array[i] < minNum)

minNum = array[i];

cout << array[i] << " ";

}

cout << endl;

}

cardsDisplay(n, allCards);

cout << minNum << endl;

vector<vector<int> > cardsUtil(4, vector<int>(13, minNum));

for(int i = 0; i < n; i ++)

{

for(int j = 0; j < 4; j ++)

{

for(int k = 0; k < 13; k ++)

{

if(cardsUtil[j][k] > 0)

{

if(allCards[i][j]->deleteByNum(k + 1))

cardsUtil[j][k] --;

}

}

}

}

cardsDisplay(n, allCards);

}

int main()

{

//testFunction();

string filename = "cards.txt";

cardsCount(filename);

return 0;

}