作业 HW1 实验报告

姓名：郑博远 学号：2154312 日期：2022 年 9 月 15 日

1. 涉及数据结构和相关背景

本次作业涉及的数据结构为线性表，包括其中的顺序表与链表。其中，第 1、2、4 题使用了顺序表解决问题，第 5 题使用了双向循环链表；第 3 题中尝试了链表与顺序表两种方式。

线性表是最常用且最简单的一种数据结构，其是由 n 个数据元素组成的有限序列。其中，线性表的顺序表指的是用一组地址连续的存储单元依次存储表的数据元素，以元素在计算机内的物理位置相邻来表示线性表中数据元素之间的逻辑关系，有着可随机存取的优点。

线性表的链式存储结构特点是用一组任意的存储单元存储线性表的数据元素（可以连续，也可以不连续），有着便于插入、删除元素的优点。

2. 实验内容

2.1 顺序表的去重

2.1.1 问题描述

输入 n 个正整数，依次插入到顺序表中。完成顺序表的去重运算，只保留顺序表中重复的第一个元素，其他均删除。

2.1.2 基本要求

输入：第一行为一个正整数 n，表示顺序表中元素的个数；

第二行为 n 个正整数 a_i，表示表中的元素。

输出：输出为一行，打印去重后顺序表中的所有元素。

2.1.3 数据结构设计

```
typedef int ElemType;

struct SqList{
    ElemType *beg;        //指向数据元素的基地址
    int length;           //顺序表的当前长度
    int listsize;     //顺序表的最大容量
};
```

2.1.4 功能说明（函数、类）

```
/*
*Status InitList(SqList& L)
* @param L 需要初始化的顺序表的指针
*/
Status InitList(SqList& L)  //空顺序表的建立


/*
*Status DestroyList(SqList& L)
* @param L 顺序表的指针
*/
Status DestroyList(SqList& L)  //顺序表的销毁


/*
*int ListLength(SqList L)
* @param L 顺序表的指针
*/
int ListLength(SqList L)       //求顺序表的长度


/*
* Status ListInsert(SqList& L, int i, ElemType e)
* @param L 顺序表的指针
* @param i 插入元素的位置
* @param e 插入元素的值
*/
Status ListInsert(SqList& L, int i, ElemType e)
//在顺序表中插入元素


/*
* Status ListDelete(SqList& L, int i, ElemType& e)
* @param L 顺序表的指针
* @param i 删除元素的位置
```

```
* @param e 删除元素的值
*/
Status ListDelete(SqList& L, int i, ElemType& e)
//在顺序表中删除元素


/*
* Status GetElem(SqList& L, int i, ElemType& e)
* @param L 顺序表的指针
* @param i 要查询元素的位置
* @param e 要查询元素的值
*/
Status GetElem(SqList& L, int i, ElemType& e)
//在顺序表查询元素的值
```

2.1.5 调试分析

调试过程中遇到重复元素删除不全、删除元素位置错位的情况。原因是在查重的时候采取 $O(n^2)$ 的算法，通过元素下标进行遍历；但删除元素之后，该元素后的所有元素会产生下标的偏移，因此在删除元素后要让正在遍历的下标自减。解决这个问题的另一种方式是，从顺序表末尾元素往前遍历，这样删除的元素不会扰动之前元素的位置，也就不会造成删除错位的问题。

2.1.6 总结和体会

在本题中主要体会了顺序表的基本写法，包括建立、销毁、插入、删除等基础函数的编写。题目比较基础，没有太多难点，主要考察对顺序表的基本掌握。调试过程中遇到的小错误耽搁了时间，希望之后在写代码时能够更加细致。

2.2 学生信息管理

2.2.1 问题描述

通过建立顺序表，存储学生信息（包含姓名与学号），并实现如下功能：

1. 根据指定学生个数，依次输入学生信息；

2. 输入新的学生信息，插入到表中指定的位置；

3. 删除指定位置的学生信息；

4. 根据姓名或学号进行查找，返回对应学生的信息；

5. 统计表中学生个数并打印。

2.2.2 基本要求

输入第一行是一个整数 n，代表学生总数；接下来 n 行包含学生的学号、姓名，用空格分割；接下来是若干行对顺序表的操作，包含如下几种：

1. insert i 学号 姓名：表示在第 i 个位置插入学生信息，若 i 位置不合法呃、则输出-1，否则输出 0；

2. remove j：表示删除第 j 个元素，若位置错误则输出-1，否则输出 0；

3. check name 姓名 y：查找姓名 y 在顺序表中是否存在，若存在则输出其位置序号及学号、姓名，若不存在则输出-1；

4. check no 学号 x：查找学号 x 在顺序表中是否存在，若存在则输出其位置序号及学号、姓名，若不存在则输出-1；

5. end：输入结束，程序输出学生总人数并退出。

2.2.3 数据结构设计

```
typedef struct info{
    char id[32];          //学生学号
    char name[32];        //学生姓名
}ElemType;


struct SqList{
    ElemType *beg;        //指向数据元素的基地址
    int length;           //顺序表的当前长度
    int listsize;         //顺序表的最大容量
};
```

2.2.4 功能说明（函数、类）

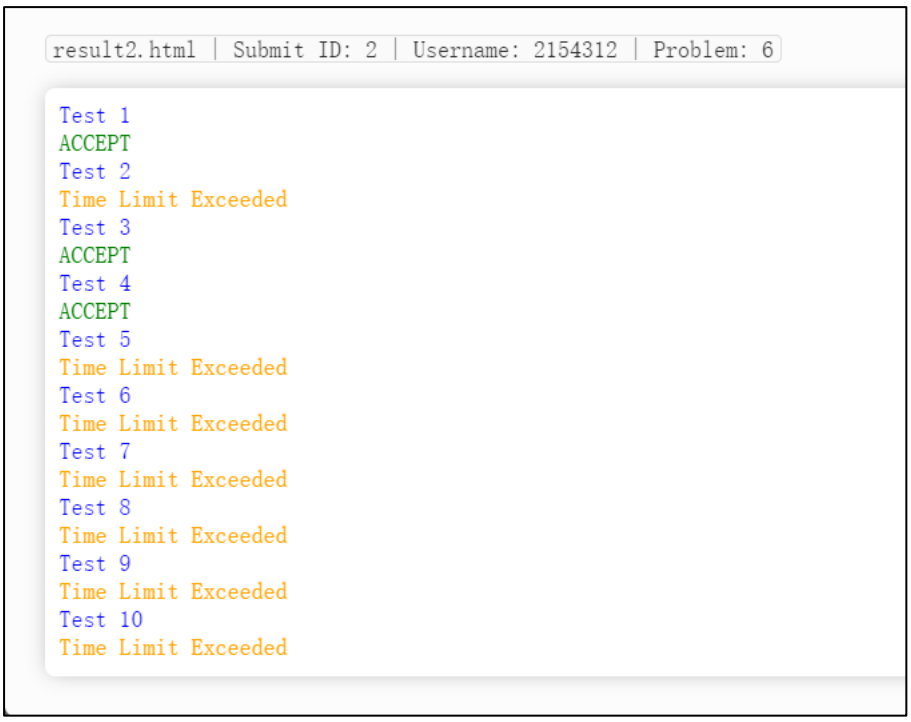
```
/*
```

```
*Status InitList(SqList& L)
* @param L 需要初始化的顺序表的指针
*/
Status InitList(SqList& L)  //空顺序表的建立


/*
*Status DestroyList(SqList& L)
* @param L 顺序表的指针
*/
Status DestroyList(SqList& L)  //顺序表的销毁


/*
*int ListLength(SqList L)
* @param L 顺序表的指针
*/
int ListLength(SqList L)       //求顺序表的长度


/*
* Status ListInsert(SqList& L, int i, ElemType e)
* @param L 顺序表的指针
* @param i 插入元素的位置
* @param e 插入元素的值
*/
Status ListInsert(SqList& L, int i, ElemType e)
//在顺序表中插入元素


/*
* Status ListDelete(SqList& L, int i, ElemType& e)
* @param L 顺序表的指针
* @param i 删除元素的位置
* @param e 删除元素的值
*/
Status ListDelete(SqList& L, int i, ElemType& e)
//在顺序表中删除元素


/*
* Status GetElem(SqList& L, int i, ElemType& e)
* @param L 顺序表的指针
* @param i 要查询元素的位置
```

```
* @param e 要查询元素的值
*/
Status GetElem(SqList& L, int i, ElemType& e)
//在顺序表查询元素的值
```

2.2.5 调试分析

1. 没有注意到题面中要求学号必须使用字符串输入，而使用了 int 类型来存储。在许多测试点中，学号包含了非数字的字符，这回导致 cin 读入错误，从而使程序死循环，最终导致 TLE 如下图。使用 char*字符串来存储学号便可以解决该问题，提醒我要注意认真阅读题面的信息；



2. 使用 string 来存储学生姓名，同时使用 malloc 来进行动态内存申请，在写入 string 时出现错误。原因是 malloc 并不会调用构造函数，因此 string 类没有被构造，导致错误。为了输入输出效率改为 char*，解决问题。

2.2.6 总结和体会

与上题类似，本题涉及到的顺序表也较为基础，包括了建立、销毁、插入、删除等基础操作。不同的是，顺序表数据元素并非已有的数据类型，而是自己定

义的结构体。使用 malloc 申请空间时，不会调用对应类型中类的构造函数，delete 释放空间时也不会调用析构函数；因此，若结构体中有类（如 string）则应注意使用 C++方式的 new 和 delete 来申请和释放空间。同时，学号要用字符串存储是本题的一个易错点；若用 int 存储会因为输入错误程序卡死而 TLE。

2.3 一元多项式的相加和相乘

2.3.1 问题描述

用一个长度为 m 且每个元素有两个数据项（系数项和指数项）的线性表((p1,e1),(p2,e2),...,(pm,em))可以唯一地表示多项式。读入两个多项式，按上述方式存储，计算并输出它们相加和相乘的运算结果。

2.3.2 基本要求

输入的第一行是一个整数 m，表示第一个一元多项式的长度；

第二行有 2m 项（p_i, e_i），中间以空格分割，表示第一个多项式各项的系数和指数；

第三、四行与前两行类似，表示第二个多项式的长度与各项系数、指数；

第五行是一个整数，若为 0 则执行加法运算并输出结果，若为 1 则执行乘法运算并输出结果，若为 2 则输出一行加法结果和一行乘法结果。

输出运算后的多项式链表时，要求按指数从小到大排列；当运算结果为 0 0 时，不输出。

2.3.3 数据结构设计

```
typedef struct item {
    int p;     //项的系数
    int e;     //项的指数
}ElemType;

typedef struct LNode {
```

```
    ElemType data;        //存放的数据
    LNode* next;      //直接后继的指针
}*Linklist;
```

2.3.4 功能说明（函数、类）

```
/*
*Status CreateList(SqList& L, int n)
* @param L 需要初始化的链表的指针
* @param n 链表初始元素的个数
*/
Status CreateList(SqList& L, int n)  //建立有 n 个元素的链表

/*
*Status DestroyList(SqList& L)
* @param L 需要销毁的链表
*/
Status DestroyList(SqList& L)  //链表的销毁

/*
* Status ListInsert(SqList& L, int i, ElemType e)
* @param L 执行操作的链表头指针
* @param i 插入元素的位置
* @param e 插入元素的值
*/
Status ListInsert(SqList& L, int i, ElemType e) //在链表中插入元素

/*
* Status ListDelete(SqList& L, int i, ElemType& e)
* @param L 执行操作的链表头指针
* @param i 删除元素的位置
* @param e 删除元素的值
*/
Status ListDelete(SqList& L, int i, ElemType& e) //在链表中删除元
素

/*
* Status GetElem(SqList& L, int i, ElemType& e)
* @param L 执行操作的链表头指针
* @param i 要查询元素的位置
* @param e 要查询元素的值
*/
Status GetElem(SqList& L, int i, ElemType& e) //在链表查询元素的值
```

```
/*
* Status AddPolynomial(Linklist L1, Linklist L2, Linklist& sum,
int mulp = 1, int mule = 0)
* @param L1 要操作的第一个链表头指针
* @param L2 要操作的第二个链表头指针
* @param sum 存入和多项式结果的链表头指针
* @param mulp 将L2的每一项乘上固定的系数，用于后续乘法，加法操作缺省1
* @param mule 将L2的每一项加上固定的指数，用于后续乘法，加法操作缺省0
*/
Status AddPolynomial(Linklist L1, Linklist L2, Linklist& sum, int
mulp = 1, int mule = 0)     //将多项式 L1,L2 相加 和存入 sum 中

/*
* Status MulPolynomial(Linklist L1, Linklist L2, Linklist& ans)
* @param L1 要操作的第一个链表头指针
* @param L2 要操作的第二个链表头指针
* @param ans 存入积多项式结果的链表头指针
*/
Status  MulPolynomial(Linklist  L1,  Linklist  L2,  Linklist&  ans)

//将多项式 L1,L2 相乘 积存入 ans 中
```

2.3.5 调试分析

用链表实现的多项式乘法在数据较大的情况下只能得到 100 分，之后的 5 个测试点会 TLE。优化前乘法采用逐项相乘再相加的方式，优化成 p 的逐项与多项式 q 相乘累加后有所进步，但由于链表单次查询的时间复杂度是 O(n)仍然无法通过测试样例。考虑到数据中项数多，空出的项少，使用顺序表在查询速度和所占空间上都不比链表逊色，最后用顺序表通过了所有测试点。

2.3.6 总结和体会

在本题中，我掌握了通过链表存储和访问数据以及各种基本函数的写法；同时，在调试过程中也巩固了对书本中多项式的加法、乘法的掌握。通过对链表和顺序表两种方式实现的对比，我更清楚的了解到了它们在各个方面的优劣。

2.4 求级数

2.4.1 问题描述

输入 N，输出级数 $A+2A^2+3A^3+\cdots+NA^N$ 的整数值。

2.4.2 基本要求

输入为一行，包含 N 和 A 的值（1<=N<=150，0<=A<=15）。

输出一行，为级数 $A+2A^2+3A^3+\cdots+NA^N$ 的整数值。

2.4.3 数据结构设计

用若干个大小为 MAXN 的数组存放大整数（低位在前，高位在后）。

2.4.4 功能说明（函数、类）

```
/*
* void print(int* c)
* @param c 要打印的数组
*/
void print(int* c)  //打印数组 c 所表示的数字


/*
* void add(int* a, int* b)
* @param a 要操作的数组一
* @param b 要操作的数组二
*/
void add(int* a, int* b)  //将数组 a 和 b 相加，结果放入数组 a 中

/*
* void mul(int* a, int* b)
* @param a 要操作的数组一
* @param b 要操作的数组二
*/
void mul(int* a, int* b)  //将数组 a 和 b 相乘，结果放入数组 b 中
```

2.4.5 调试分析

有先前大整数乘法的基础，写的比较顺利，没有遇到问题。

2.4.6 总结和体会

在 HW0 高精度乘法的基础上修改且链表需额外花费空间存放指针，因此没

有选择链表的方式；考虑到相乘的位数在可接受范围内，也没有通过动态内存申请顺序表。主要巩固一下了对高精度加法和乘法的掌握，总体完成的很顺畅。

2.5 扑克牌游戏

2.5.1 问题描述

对于一个扑克牌堆，定义以下 4 种操作命令：

1. 添加（Append）：添加一张扑克牌到牌堆的底部；

2. 抽取（Extract）：从牌堆中抽取某种花色的所有牌，按照编号从小到大进行排序，并放到牌堆的顶部；

3. 反转（Revert）：使整个牌堆逆序；

4. 弹出（Pop）：如果牌堆非空，则除去牌堆顶部的第一张牌，并打印该牌的花色和数字；如果牌堆为空，则打印 NULL。

初始时牌堆为空。输入 n 个操作命令（1 ≤ n ≤200），执行对应指令。

所有指令执行完毕后，按照从牌堆顶到牌堆底的顺序打印牌堆中所有牌花色和数字。如果牌堆为空，则打印 NULL。

2.5.2 基本要求

输入的第一行是一个整数 n，表示命令的数量；接下来有 n 行输入，每一行输入一个命令。

输出有若干行，每次收到 Pop 指令后输出一行（花色和数子或 NULL），最后将牌堆中的牌从牌堆顶到牌堆底逐一输出，若牌堆为空则输出 NULL。

2.5.3 数据结构设计

选择带头节点的双向循环链表来存储扑克牌堆，结构如下：

```
typedef struct Poker {
    char type[128];      //扑克牌花色
```

```
    char num[4];      //扑克牌点数
}Elemtype;
Elemtype vec[256];

typedef struct DuLNode {
    Elemtype data;              //该节点存放的扑克牌信息
    DuLNode* prior, * next; //该节点的直接前驱、后继的指针
}* DuLinkList;
```

2.5.4 功能说明（函数、类）

```
/*
*Status InitList(DuLinkList& L)
* @param L 需要初始化的链表的指针
*/
Status InitList(DuLinkList& L)  //空循环链表的建立


/*
*Status DestroyList(DuLinkList& L)
* @param L 循环链表的指针
*/
Status DestroyList(DuLinkList& L)  //循环链表的销毁

/*
*Status PrintList(DuLinkList L, bool to_next)
* @param L 循环链表的指针
* @param to_next 遍历方式向后继还是向前驱
*/
Status PrintList(DuLinkList L, bool to_next)  //循环链表的打印


/*
* Status InsertItem(DuLinkList& L, ElemType e, bool to_next)
* @param L 循环链表的头指针
* @param e 插入元素的值
* @param to_next 遍历方式向后继还是向前驱
*/
Status InsertItem(DuLinkList& L, ElemType e, bool to_next)
 //在链表头部插入元素


/*
* Status PopItem(DuLinkList& L, ElemType e, bool to_next)
* @param L 循环链表的头指针
```

```
* @param e 抽出元素的值
* @param to_next 遍历方式向后继还是向前驱
*/
Status PopItem(DuLinkList& L, ElemType e, bool to_next)
//在链表头部删除元素


/*
* Status ExtractItem(DuLinkList& L, char* type, bool to_next)
* @param L 循环链表的头指针
* @param type 当前Extract的花色
* @param to_next 遍历方式向后继还是向前驱
*/
Status ExtractItem(DuLinkList& L, char* type, bool to_next)
//对链表做题目所述的 Extract 操作
```

2.5.5 调试分析

问题 1：起初用 char 变量存放，不能适应牌的数字为 10 的情况；因此，修改之后改为字符串方式存储解决问题，能适应更复杂的花色；

问题 2：在做“Extract”操作时，有时会出现不能将牌堆中的所有牌完全取出的情况。原因是“Revert”操作前后，遍历双向循环链表的方式会发生改变，而有一处仅写了向后继方向遍历，导致遍历提前终止。

2.5.6 总结和体会

在本题中，我使用了双向循环链表来处理牌堆的“Revert”操作，即：用一个 bool 变量 to_next 来记录遍历链表、插入元素等操作时是向后继方向还是向前驱方向，并在“Revert”将 to_next 取反。由于本题对链表的操作比较固定，我没有统一调用 ListDelete、ListInsert，选择了将不同的操作分别封装成函数。除了在上文提到的地方调试较久以外，总体的编程过程比较顺利。

3. 实验总结

这次作业中，我巩固了对线性表这一数据结构的了解与使用，在不同的题目

中分别应用了顺序表、单向链表、双向循环链表来解决问题，操作还算熟练。在“一元多项式的相加和相乘”一题中，我还比较了顺序表与链表这两种线性表的异同，体会了二者在插入、删除、查找等方面的特点，以及时间复杂度与空间复杂度在不同情况下的优劣。

在本次作业的几题中，我熟练地掌握了线性表的顺序表与链式结构的插入、删除、查找等功能函数；同时，我在几道题目中体会了线性表的经典应用，总体的编程过程比较顺畅。

总的来说，通过本次作业，我加深了对线性表的理解，也复习了各种基础操作。我希望通过这次作业的练习在未来能更加熟练掌握使用线性表。