作业 HW1 实验报告

姓名：何正潇 学号：1950095 日期：2021 年 10 月 13 日

1. 涉及数据结构和相关背景

本单元的数据结构主要涉及的都是线性结构，主要题目涉及的是顺序表和链表数据结构的考察。且可以使用静态和动态的内存申请进行编写程序。

2. 实验内容

2.1 题目：1-1 学生信息管理

2.1.1 问题描述

此题其实是应用一些顺序表的基本操作。顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。 顺序表的基本操作，包括顺序表的创建，第 i 个位置插入一个新的元素、删除第 i 个元素、查找某元素、顺序表的销毁。

2.1.2 基本要求

定义一个包含学生信息（学号，姓名）的的 顺序表，使其具有如下功能：(1) 根据指定学生个数，逐个输入学生信息；(2) 给定一个学生信息，插入到表中指定的位置；(3) 删除指定位置的学生记录；(4) 分别根据姓名和学号进行查找，返回此学生的信息；(5) 统计表中学生个数。

2.1.3 数据结构设计

针对顺序表的特性及具体实验题目的要求，定义，

- 常量定义

```
#define OVERFLOW -2 //当动态生成更大的顺序表时，如果连续内存不够，导致失败
#define INSERT_NUM 120 //每次动态生成顺序表后，预设可以再插入的元素数
```

- 结构定义

针对题目要求，定义顺序表的元素结构，

```
struct s_element
{
    char id[10];
    char name[20];
};
```

2.1.4 功能说明（函数、类）

针对题目要求的基本顺序表类声明，

```
class CSquential_List
```

```
{
public:
    CSquential_List(const int num);
    ~CSquential_List();

    int Insert_Elem(const int pos, const s_element elem);
    int Remove_Elem(const int pos);
    int Check_Name(char* num, const char* name);
    int Check_id(char* name, const char* id);
    int Get_Num_Elems() { return m_num_elems; }
private:
    s_element* m_elems;
    int m_size;
    int  m_num_elems;
};
```

考虑到顺序表的内存的特殊性（连续存储区域），类的构造函数、析构函数、Insert_Elem 函数中，调用基本的 C 语言的动态内存分配、内存释放、和内存重分配的库函数 malloc、free 和 realloc。

2.1.5 调试分析（遇到的问题和解决方法）

由于我刚转入计算机系的大数据专业，第一次接触 OJ 系统，因此，我通过有意在不同的编译器、动态内存分配的漏洞、一些输入数据超越边界的处理漏洞等，逐步了解 OJ 测试系统。

2.1.6 总结和体会

- 收获

1） 了解 OJ 系统

2） 编写顺序表基本功能类申明和类函数的实现

- 难点和易错点

1） 程序运行过程中，动态内存重分配

2） 数据边界判定（即插入时的顺序表的合法位置）

2.2 题目二：1-2 有序表的合并问题

2.2.1 问题描述

已知线性表 La 和 Lb 的元素按值非递减有序排列，现将 La 和 Lb 归并成一个新的线性有序表 Lc，且 Lc 中的数据元素仍然按照值非递减有序排列。输入 La 和 Lb 值的时候可以非有序，输出合并后的结果。

2.2.2 基本要求

对 La 和 Lb 进行排序，然后采用归并排序算法输出 Lc。

2.2.3 数据结构设计

```
typedef struct numbers
{
    Elemtype* elem;
    int length;
```

```
    int listsize;
} SQlist;
```

进行结构体定义，elem 用来申请保存顺序表的空间，length 用来记录顺序表的长度，listsize 用来记录现在顺序表申请空间的大小。

2.2.4 功能说明（函数、类）

```
#include <iostream>
#include<string.h>
using namespace std;
#define LIST_INIT_SIZE 10
#define LISTINCREMENT 10
typedef int Elemtype;
typedef int Status;
typedef struct numbers
{
    Elemtype* elem;
    int length;
    int listsize;
} SQlist;
//顺序表初始化程序
Status Initlist_sq(SQlist& L)
{
    L.elem = (Elemtype*)malloc(LIST_INIT_SIZE * sizeof(Elemtype));
    if (!L.elem)
        return -1;
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    int i = 1;
    while (1)
    {
        int temp;
        cin >> temp;
        if (temp == 0)
            break;
        if ((L.length+1) >= L.listsize)
        {
            Elemtype* newbase;
            newbase = (Elemtype*)realloc(L.elem, (LISTINCREMENT+L.listsize) * sizeof(Elemtype));
            if (!newbase)
                return -1;
            L.elem = newbase;
            L.listsize = L.listsize + LISTINCREMENT;
        }
        L.elem[i++] = temp;
```

```
        L.length++;
    }
    return 0;
}
Status Initlist_sq_(SQlist& L)
{
    L.elem = (Elemtype*)malloc(LIST_INIT_SIZE * sizeof(Elemtype));
    if (!L.elem)
        return -1;
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    if ((L.length + 1) >= L.listsize)
    {
        Elemtype* newbase;
        newbase = (Elemtype*)realloc(L.elem, (LISTINCREMENT + L.listsize) * sizeof(Elemtype));
        if (!newbase)
            return -1;
        L.elem = newbase;
        L.listsize = L.listsize + LISTINCREMENT;
    }
    return 0;
}
//顺序表插入程序
Status ListInsert(SQlist& L, int i, Elemtype e)
{
    if (i<1 || i>(L.length+1) )
        return -1;
    if ((L.length + 1) >= L.listsize) {
        Elemtype* newbase;
        newbase = (Elemtype*)realloc(L.elem, (LISTINCREMENT + L.listsize) * sizeof(Elemtype));
        if (!newbase)
            return -1;
        L.elem = newbase;
        L.listsize = L.listsize + LISTINCREMENT;
    }
    Elemtype* q = &(L.elem[i ]);
    for (Elemtype* p = &(L.elem[L.length ]); p >= q; p--) {
        memcpy(p + 1, p, sizeof(Elemtype));
    }
    memcpy(q, &e, sizeof(Elemtype));
    L.length++;
    return 0;
}
//顺序表 La 和 Lb 的归并排序程序
```

```
void Mergelist(SQlist La, SQlist Lb, SQlist& Lc)
{
    int i=1, j = 1;
    int k = 0;
    int La_len = La.length, Lb_len = Lb.length;
    while ((i <= La_len) && (j <= Lb_len))
    {
        int ai = La.elem[i ], bj = Lb.elem[j ];
        if (ai <= bj)
        {
            ListInsert(Lc, ++k, ai);
            ++i;
        }
        else
        {
            ListInsert(Lc, ++k, bj);
            ++j;
        }
    }
    while (i <= La_len)
    {
        int ai = La.elem[i ];
        i++;
        ListInsert(Lc, ++k, ai);
    }
    while (j <= Lb_len)
    {
        int bj = Lb.elem[j];
        j++;
        ListInsert(Lc, ++k, bj);
    }
}
//顺序表内部的冒泡排序程序
void Line_up(SQlist& La)
{
    for (int i = 0; i < La.length-1; i++)
    {
        for (int j = 1; j < La.length-i; j++)
        {
            if (La.elem[j] >= La.elem[j + 1])
            {
                int t = La.elem[j];
                La.elem[j] = La.elem[j + 1];
                La.elem[j + 1] = t;
```

```
            }
        }
    }
}
int main()
{
    SQlist a, b,c;
    Initlist_sq(a), Initlist_sq(b);
    Line_up(a), Line_up(b);
    Initlist_sq_(c);
    Mergelist(a, b, c);
    for (int i = 1; i < c.length+1; i++)
        cout << c.elem[i]<<" ";
    return 0;
}
```

2.2.5 调试分析（遇到的问题和解决方法）

主要的难点在于线性表的排序和归并算法设计，我在这里使用的是冒泡排序方法和比较经典的归并排序算法。

2.2.6 总结和体会

通过这道题主要是体验归并排序的算法以及线性表的基本操作实践，难度总体来说不大。

2.3 题目三：1-3 扑克牌游戏

2.3.1 问题描述

扑克牌有 4 种花色：黑桃（Spade）、红心（Heart）、梅花（Club）、方块（Diamond）。每种花色有 13 张牌，编号从小到大为：A,2,3,4,5,6,7,8,9,10,J,Q,K。

2.3.2 基本要求

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。如命令“Append Club Q”表示添加一张梅花 Q 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。如命令“Extract Heart”表示抽取所有红心牌，排序之后放到牌堆的顶部；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打印牌堆中所有牌花色和数字（从牌堆顶到牌堆底），如果牌堆为空，则打印 NULL。注意：每种花色和编号的牌数量不限。

2.3.3 数据结构设计

根据题目描述和基本要求，此题是线性表中链表的基本应用。考虑具体题目

- 常量定义

```
#define Spade 1
```

```
#define Heart 2
#define Club 3
#define Diamond 4
#define A 1
#define J 11
#define Q 12
#define K 13
```

- 结构定义

```
struct s_card
{
    int  data_1; //1: spade; 2 heart; 3: club; 4: diamond
    int  data_2; //1: A; 2-10; 11: J; 12: Q; 13: K
    s_card(int data_1_, int data_2_) : data_1(data_1_), data_2(data_2_) {}
};

struct s_node
{
    s_card    card;
    s_node* next;
    s_node* prev;
    s_node(s_card card_) : card(card_), next (0), prev(0) {}
};
```

2.3.4 功能说明（函数、类）

- 针对题目的双向链表声明,

```
class CLink_List
{
public:
    CLink_List();
    ~CLink_List();
    void INS_Node_Bef(s_node* inA, s_node* inB); //insert inB node, before the inA node of the
                                                 //CLinkList object
    void INS_Node_Beh(s_node* inA, s_node* inB); //insert inB node, behind the inA node of the
                                                 //CLinkList object
    void DEL_Node(s_node* inNode); //delete a node in the object of CLinkList
    void RM_Node(s_node* inNode); //remove a node in the object of CLinkList, but the node is not
                                  //deleted
    bool Get_Node(s_node* outNode); //release the node after head
    void RET_Node(s_node* inNode); //return a node into link list, the position after head

    bool Pop_Node(s_node* &outNode); //pop the node before tail
    void Reverse(); //reverse link list

    s_node* Get_Head() { return m_head; }
```

```
    s_node* Get_Tail() { return m_tail; }
    int Get_Len() { return m_len; }
private:
    s_node* m_head;
    s_node* m_tail;
    int  m_len;
};
```

上面声明中的注释简述了每个类函数的功能

- 特别的类函数定义

针对"Revert"命令，类函数 Rerverse()的实现，

```
void CLink_List::Reverse()
{
    s_node* tmp1, *tmp2, *node;
    if (m_len > 0)
    {
        node = m_head;
        while (node)
        {
            tmp1 = node->next;
            tmp2 = node->next;
            node->next = node->prev;
            node->prev = tmp2;
            node = tmp1;
        }
        tmp1 = m_head;
        m_head = m_tail;
        m_tail = tmp1;
    }
}
```

- "Extract"命令的函数实现，

```
void extract(int color, CLink_List* link_list)
{
    s_node* node;
    s_node** node_array;
    int* card_num;
    int num_extract, cnt;

    num_extract = 0;
    node = link_list->Get_Head()->next;
    while (node->card.data_1 > 0)
    {
        if (node->card.data_1 == color)
        {
```

```
                num_extract++;
            }
            node = node->next;
        }
        if(num_extract > 0)
        {
            node_array = new s_node*[num_extract];
            cnt = 0;
            node = link_list->Get_Head()->next;
            while (node->card.data_1 > 0)
            {
                if (node->card.data_1 == color)
                {
                    node_array[cnt] = node;
                    link_list->RM_Node(node);
                    cnt++;
                }
                node = node->next;
            }
            bubbleSort(node_array, num_extract);
            for (int i = 0; i < num_extract; i++)
                link_list->INS_Node_Bef(link_list->Get_Tail(), node_array[i]);
            delete[] node_array;
        }
    }
```

及 extract 函数中调用的 bubbleSort 函数，

```
void bubbleSort(s_node **node_array, int len)
{
    s_node *tmp;
    int i, j;
    for (i = 0; i < len - 1; i++)
    {
        for (j = 0; j < len - 1 - i; j++)
        {
            if (node_array[j]->card.data_2 < node_array[j + 1]->card.data_2)
            {
                tmp = node_array[j];
                node_array[j] = node_array[j + 1];
                node_array[j + 1] = tmp;
            }
        }
    }
}
```

2.3.5 调试分析（遇到的问题和解决方法）

基本没有遇到太多问题。

2.3.6 总结和体会

- 收获

编写双向链表的类声明和类函数的实现

- 难点和易错点

1） 实现”Revert”命令的链表类中的 Reverse 类函数的实现

2） 实现“Extract“命令的 extract 函数的实现

2.4 题目四一元多项式的相加和相乘

2.4.1 问题描述

输入两个线性表，进行线性表中保存的一元多项式的相加和相乘，同时按照指数从小到大的顺序进行输出

2.4.2 基本要求

输入一元多项式长度，输入线性表 La 的系数以及指数，输入第二个一元多项式的长度，输入线性表 Lb 的系数以及指数。根据输入的命令，进行一元多项式的乘法和加法运算。

2.4.3

数据结构设计

```
typedef struct poly {
    int p;//系数
    int coe;//指数
}poly;

struct list {
    struct poly* Elem;
    int length;
};
```

设定数据结构 poly 记录一元多项式的次数和系数，同时结构体 list 保存线性表的首地址，同时也记录线性表的长度。以便后面的乘法和加法运算。

2.4.4 功能说明（函数、类）

```
/*1950095 大数据 何正潇*/
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
/*
注释:此处用顺序表较为简单,因此没有使用链表,基本使用的就是归并排序的思想,与教科书例题代码相似。
*/
typedef struct poly {
    int p;//系数
    int coe;//指数
}poly;

struct list {
    struct poly* Elem;
    int length;
};
/*冒泡排序*/
void sort(list& L) {
    poly k;
    for (int i = 1; i <= L.length - 1; i++)
        for (int j = 1; j <= L.length - i; j++)
            if (L.Elem[j].coe > L.Elem[j + 1].coe)
            {
                k = L.Elem[j];
                L.Elem[j] = L.Elem[j + 1];
                L.Elem[j + 1] = k;
            }
}

/*输出多项式*/
void print(list& L) {
    for (int i = 1; i <= L.length; i++) //输出
        if (L.Elem[i].p != 0) //系数不为0输出
            cout << L.Elem[i].p << " " << L.Elem[i].coe << " ";
    cout << endl;
}
/*归并排序*/
void add(list& L1, list& L2, list& L3) {
    sort(L1);
    sort(L2);
    int i = 1, j = 1, k = 1;
    while (i <= L1.length && j <= L2.length) {
        if (L1.Elem[i].coe < L2.Elem[j].coe)
```

```
                L3.Elem[k++] = L1.Elem[i++];
            else if (L1.Elem[i].coe > L2.Elem[j].coe)
                L3.Elem[k++] = L2.Elem[j++];
            else {
                L3.Elem[k].coe = L1.Elem[i].coe;
                L3.Elem[k++].p = L1.Elem[i++].p + L2.Elem[j++].p;
            }
        }
        while (i <= L1.length)
            L3.Elem[k++] = L1.Elem[i++];
        while (j <= L2.length)
            L3.Elem[k++] = L2.Elem[j++];
        L3.length = k - 1;
    }
    /*项数Delete*/
    void Delete(list& L, int position)
    {
        for (int i = position; i <= L.length; i++)
            L.Elem[i] = L.Elem[i + 1];
        L.length--;
    }
    /*乘法*/
    void multiply(list& L1, list& L2, list& L3) {
        int i, j, k = 1;
        for (i = 1; i <= L1.length; i++)
            for (j = 1; j <= L2.length; j++)
            {
                L3.Elem[k].p = L1.Elem[i].p * L2.Elem[j].p;
                L3.Elem[k].coe = L1.Elem[i].coe + L2.Elem[j].coe;
                k++;
            }
        L3.length = L1.length * L2.length;
        sort(L3);
        for (int i = 1; i < L3.length; i++) {
            if (L3.Elem[i].coe == L3.Elem[i + 1].coe)
            {
                L3.Elem[i].p += L3.Elem[i + 1].p;
                Delete(L3, i + 1);
                i--;//此处需要注意
            }
        }
    }

    int main()
```

```
{
    list L1, L2, L3;
    L1.Elem = (poly*)malloc(10000 * sizeof(poly));
    L2.Elem = (poly*)malloc(10000 * sizeof(poly));
    L3.Elem = (poly*)malloc(10000 * sizeof(poly));
    L3.length = 0;
    cin >> L1.length;
    for (int i = 1; i <= L1.length; i++)
        cin >> L1.Elem[i].p >> L1.Elem[i].coe;//输入
    cin >> L2.length;
    for (int i = 1; i <= L2.length; i++)
        cin >> L2.Elem[i].p >> L2.Elem[i].coe;//输入
    int order;
    cin >> order;//输入命令
    if (order == 0) {
        add(L1, L2, L3);
        print(L3);
    }
    else if (order == 1) {
        multiply(L1, L2, L3);
        print(L3);
    }
    else if (order == 2) {
        add(L1, L2, L3);
        print(L3);
        multiply(L1, L2, L3);
        print(L3);
    }
    free(L1.Elem);
    free(L2.Elem);
    free(L3.Elem);
    return 0;
}
```

2.4.5 调试分析（遇到的问题和解决方法）

本题难度相对来说较低，但是综合性较高，其中包含了归并排序，冒泡排序，链表的合成等多个知识点。

2.4.6 总结和体会

通过这道题主要是体验归并排序的算法以及线性表的基本操作实践，我本来打算在这里使用链表进行操作，但是发现实际上在本题中链表操作并不简单，于是改用顺序表，调试一次以后通过本次程序。

2-5 级数相加

2.5.1 问题描述

若干行，在每一行中给出整数 N 和 A 的值，（1<=N<=150，0<=A<=15），计算得到级数运算的结果。

2.5.2 基本要求

由于该题中涉及大数的乘法和加法，所以需要用数组保存每一位的数字。才能保证大数字的计算值正确。

2.5.3 数据结构设计

```
int answer[400]
```

该题主要考察的是简单顺序表的应用，所以不需要复杂的数据结构。

2.5.4 功能说明（函数、类）

```
        /*1950095 大数据 何正满*/
        #include <iostream>
        #include <cstring>
        using namespace std;

        /*函数功能：实现高精度乘法*/
        void mul(int num1[400], int answer[400], int num2, int& length,int&
length_)
        {
            int i = 0;
            int residue = 0;
            for (i = 0; i < length; i++)
            {
                int temp = num1[i];
                answer[i] = (residue + num2 * (num1[i])) % 10;
                residue = (residue + num2 * (temp)) / 10;

            }
            while (residue)
            {

                answer[i] = residue % 10 ;
                residue = residue / 10;
                length++;
                i++;
            }
            length_ = length;

        }
        /*函数功能：实现高精度加法*/
```

```
        void add(int num1[400], int num2[400], int answer[400], int& length,
int& length_)
        {
            int residue = 0;
            int i = 0;
            for (i = 0; i < length_; i++)
            {
                int temp = num2[i];
                answer[i] = (residue + num2[i] + num1[i]) % 10;
                residue = (residue + num1[i] + temp) / 10;
            }
            while (residue)
            {

                answer[i] = residue % 10;
                residue = residue / 10;
                length_++;
                i++;
            }

        }
        int main()
        {
            int N, A;
            cin >> N >> A;
            int length = 0;
            int length_ = 0;
            if (A < 10 && A >= 0)
                length_ = 1;
            else
                length_ = 2;
            length = length_;
            int num1[400];
            int num2[400];//没有必要，可以忽略
             int answer[400];
             memset(answer, 0, 1600);
             memset(num1, 0, 1600);
             num1[0] = A % 10, num1[1] = A / 10 % 10, num1[2] = A / 10 / 10 %
10;
             for (int i = 1; i <= N; i++)
             {
                 if (A < 10 && A >= 0)
                     length = 1;
                 else
```

```
                    length = 2;
                if (i == 1)
                {
                    if (length_ == 1)
                        answer[0] = A;
                    else
                    {
                        answer[0] = A % 10;
                        answer[1] = A / 10 % 10;
                    }
                }
                else
                {
                    for (int j = 1; j < i; j++)
                    {
                        mul(num1, num1, A, length,length_);
                    }
                    mul(num1, num1, i, length, length_);
                    add(num1, answer, answer, length,length_);
                    for (int i = 3; i < length_; i++)
                        num1[i] = 0;
                    num1[0] = A % 10, num1[1] = A / 10 % 10, num1[2] = A /
10 / 10 % 10;

                }

            }
            for (int i = length_ - 1; i >= 0; i--)
                cout << answer[i];
        /*    for (int i = 1; i <= N; i++)
            {
                for (int j = 1; j <= i; j++)
                {
                    mul(num1,answer,)
                }
            }*/
            return 0;
        }
```

2.5.5 调试分析（遇到的问题和解决方法）

本题总体来说数据结构的考察难度不高，但是其中也容易遇到问题，在几个函数中需要设定临时变量 temp 记录数值，否则会导致问题。程序的核心在于每一位的运算逻辑。

2.5.6 总结和体会

本题运用了顺序表进行大数字的乘法和加法，总体趣味性较高，也为超过变量保存范围的数字运算方法提供了一种新的思路。

2. 实验总结

总体来说，本次实验对于线性表相关的基本知识进行了巩固与练习，提高了我们对于基本数据结构的了解。