作业 HW1 实验报告

姓名：刘博洋 学号：2153538 日期：2022 年 10 月 6 日

实验报告格式按照模板来即可，对字体大小、缩进、颜色等不做要求
实验报告要求在文字简洁的同时将内容表示清楚

1. 涉及数据结构和相关背景

线性表是最常用且灵活的一种数据结构，作为基本的线性结构，我们在这一章学习的线性表主要包括顺序表和链表两种，二者各有其优劣。其中顺序表的结构简单、存储效率高，结构紧凑，可以直接存取。

但是涉及插入和删除操作时需要遍历许多数据元素，复杂度较高，对于事先不知道长度的线性表数据，需要预先分配较大空间并断增加空间，操作麻烦。这时候就需要用到链表，链表虽然储存空间不连续。但是对于需要经常插入和删除的数据或问题，链表的操作更为简单，使用中也可以将链表进行改进，根据问题需要构造双向链表、循环链表等。

2. 实验内容

2.1 顺序表去重

2.1.1 问题描述

完成顺序表的去重运算，即顺序表中相同的元素只留下一个

2.1.2 基本要求

输入：

第 1 行 1 个正整数 n，表示创建的顺序表的元素个数。

第 2 行 n 个正整数 ai，表示表中元素。

其中，ai 32 位 int 范围内，n <= 50,000

输出：

1 行，去重后的所有元素

2.1.3 数据结构设计

建立顺序表，实现判断重复的操作和删除的操作，从第一个元素开始遍历，两层循环嵌套，依次比较是否重复，如有重复即执行删除操作。判断顺序表的空间是否足够，不够需要用 realloc 再分配内存。

2.1.4 功能说明（函数、类）

```
typedef struct {
	Int* elem;
	int length;
	int listsize;
}sqlist;
sqlist L;//定义顺序表类型

Status del(sqlist& L, int i)
{
	int* p;
	int* q = L.elem + L.length - 1;
	p = &(L.elem[i]);
	for (++p;p <= q;p++)
		*(p - 1) = *p;
	L.length--;
	if (L.length <= 0)
```

```
            return ERROR;
      return OK;
}

// 对顺序表中元素进行删除操作

void quchong(sqlist& L)
{
      int i, j;
      i = 0, j = 0;
      for (i = 0;i < L.length-1;i++)
      {
            for (j = i + 1;j < L.length;j++)
                  if (L.elem[j] == L.elem[i])
                  {
                        del(L, j);
                        j--;
                  }
      }
}
//嵌套调用 del 函数，循环遍历判断重复即执行删除操作
void print(sqlist& L)
{
      for (int i = 0;i < L.length;i++)
            cout << L.elem[i] << " ";
}
//print 函数顺次输出线性表的值
```

2.1.5 调试分析（遇到的问题和解决方法）

程序写完后如有报错可进行单步调试找出问题语句。整个不报错后，先分块进行调试，即先只在 main 函数中执行 del 操作，检查删除功能是否正确，再加上去重部分，利用测试数据进行测试正确性。在调试过程中，我开始遇到的问题是由于对于基本操作的编程还不太熟练，导致删除操作的边界条件出现问题，删除结果有问题，在提交阶段，答案显示正确但是得分不高，最后找到可能原因是数组越界或者空间分配不够，慢慢修改程序和调试得到最终正确的程序。

2.1.6 总结和体会

此题对算法效率要求不高，属于基本的操作。难点主要在于去重判断和删除遍历时的指针操作和退出条件。但是对于我这种刚转入的基础不太好的同学来说，基本操作的正确性和准确度还需要进一步提升，一方面在写程序的时候尽量注意一些细节，减少漏洞，另一方面在寻找错误和漏洞的过程中要保持虚心学习和耐心纠错的态度，多投入时间。

2.2 学生信息管理

2.2.1 问题描述

本问题要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。

2.2.2 基本要求

第 1 行是学生总数 n

接下来 n 行是对学生信息的描述，每行是一名学生的学号、姓名，用空格分割；(学号、

姓名均用字符串表示)
接下来是若干行对顺序表的操作：(每行内容之间用空格分隔)
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，输出-1，否则输出 0
remove j:表示删除第 j 个元素，若元素位置不合适，输出-1，否则输出 0
check name 姓名 y：查找姓名 y 在顺序表中是否存在，若存在，输出其位置序号及学号、姓名，若不存在，输出-1。
check no 学号 x：查找学号 x 在顺序表中是否存在，若存在，输出其位置序号及学号、姓名，若不存在，输出-1。
end: 操作结束，输出学生总人数，退出程序。

2.2.3 数据结构设计

建立一个结构类型 student 包含学号和姓名两个数据元素，再建立一个包含结构元素 student 的顺序表存放学生信息，插入元素时将后面元素向后移一个再插入，同时判断顺序表的空间是否足够，不够需要用 realloc 再分配内存。删除操作要删除之后将后面元素向前移，同时 length--；
遍历顺序表检查 student 中的 name 和 no 成员以查找指定的学生信息并输出。此外，对于指令的输出需要用到字符串的比较判断输入特定指令便执行特定操作。

2.2.4 功能说明（函数、类）

```
int insert(sqlist& L, int i)
{
      int a = 0;

        if (i<=0|| (i>L.length+1))
      {

          cout<<-1<<endl;
          return 0;
      }
      else cout << 0 << endl;
        if (i == L.length + 1)
        {
              L.length++;
              cin >> L.elem[i - 1].no;
              cin >> L.elem[i - 1].name;
        }
        else {
              for (int j = L.length - 1;j >= i - 1;j--)
              {
                    L.elem[j + 1] = L.elem[j];
              }
              cin >> L.elem[i - 1].no;
              cin >> L.elem[i - 1].name;
              ++L.length;
```

```
        }
    return a;

}//在指定位置执行插入操作
int remove(sqlist& L, int j)
{

    if (j<=0 || j>L.length)
    {
        cout<<-1<<endl;
        return 0;
    }
    else cout << 0 << endl;
    for (int i = j-1;i < L.length-1;i++)
    {
        L.elem[i] = L.elem[i+1];
    }
    --L.length;
    return 0;
}//删除第 j 个元素
void checkname(sqlist& L)
{
    int i;
    char name[10];
    cin >> name;
    for (i = 0;i < L.length;i++)
    {
        if (L.length == 0)
        {
            cout << -1 << endl;
            return;
        }
        if (strcmp(L.elem[i].name, name) == 0)
        {
            break;
        }
    }

    if (L.length!=0)
    {

        cout << i + 1 << " " << L.elem[i].no << " " << L.elem[i].name <<
endl;
    }
```

```
	else
		cout << -1<<endl;

}//通过名字 name 查找学生信息
void checkno(sqlist& L)
{
	int i;
	char no[10];
	cin >> no;
	for (i = 0;i < L.length;i++)
	{
		if (L.length == 0)
		{
			cout << -1 << endl;
			return;
		}
		if (strcmp(L.elem[i].no, no) == 0)
		{
			break;
		}
	}
	if (L.length == 0)
	{
		cout << -1<<endl;
		return;
	}
	if (strcmp(L.elem[i].no, no) == 0)
	{

		cout << i + 1 << " " << L.elem[i].no << " " << L.elem[i].name <<
endl;
	}
	else
		cout << -1<<endl;
}//通过学号 no 查找学生信息

do
	{

		cin >> s;
		if (s == "insert")
		{
			cin >> locate;
			insert(L, locate);
```

```
        }
        if (s == "remove")
        {
              int j;
              cin >> j;
              remove(L, j);
        }
        if (s == "check")
        {
              cin >> a;
              if (a == "name")
              {
                    checkname(L);
              }
              if (a == "no")
              {
                    checkno(L);
              }
        }

  } while (s != "end");
  cout << L.length<<endl;
//通过判断是否输入指定指令，来决定执行何种操作
```

2.2.5 调试分析（遇到的问题和解决方法）

分模块测试，分别令 main 函数中只执行查找、删除和插入操作，检验代码执行结果，再整合起来调试，遇到 bug 就进行单步调试并且观察 L 中各成员的值，找到错误点进行修改。遇到的问题有开始时插入操作执行的不够好，没有区分在表尾插入和在表中插入是两种不同的情况，导致有部分测试数据出现错误，后来增加了插入表尾的情况；还有开始在查找的时候没有考虑到是空表的情况，程序健壮性不够，随后加入判断语句，程序更加完善。

2.2.6 总结和体会

在利用线性表进行操作时，要注意内存的分配问题，否则很容易超出界限，在执行插入、删除操作时，最好要画图示来理清思路，找准临界位置和终止条件，防止出现越界。作为编程基础不够牢固的我来说，要多投入时间去调试和分析思考问题出在哪里，而不是直接上网搜现有的代码。此题让我熟悉了线性表的建立、数据输入、查找、删除等一系列基本操作。

2.3 一元多项式的相加和相乘

2.3.1 问题描述

一元多项式是有序线性表的典型应用，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))可以唯一地表示一个多项式。本题实现多项式的相加和相乘运算

2.3.2 基本要求

输入：
第 1 行一个整数 m，表示第一个一元多项式的长度
第 2 行有 2m 项，p1 e1 p2 e2 ...，中间以空格分割,表示第 1 个多项式系数和指数
第 3 行一个整数 n,表示第二个一元多项式的项数
第 4 行有 2n 项，p1 e1 p2 e2 ...，中间以空格分割,表示第 2 个多项式系数和指数
第 5 行一个整数，若为 0,执行加法运算并输出结果，若为 1，执行乘法运算并输出结果；若为 2，输出一行加法结果和一行乘法的结果。
输出：
运算后的多项式链表，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

利用线性链表存储多项式，每个节点表示一项，包含 exp（指数）和 coef（系数）两个数据元素。创建两条多项式链表，在执行加法操作时，用两个指针分别指向两链表的表头，依次移动，当两指针指向的指数相同时，执行加法操作，系数相加不为 0 则存入结果链表 result 中，如果其中一个指针先遍历完，则将另一指针后面的所有节点复制到结果链表中，则输出结果链表即为多项式加法结果。执行乘法操作时，可将乘法看成 a 多项式的每一项分别与 b 多项式相乘，具体操作为，先让 a 多项式的第一项与 b 多项式相乘存入结果链表中，然后再让第一项之后的每一项与 b 的每一项相乘，结果插入结果链表中，最后得到结果链表的值。

2.3.4 功能说明（函数、类）

```
typedef struct LNode { //结点类型
	int coef;
	int exp;
	struct LNode* next;
}*Link,*position;
typedef struct { //链表类型
	Link head, tail; /* 分别指向线性链表中的头结点和最后一个结点 */
	Link current; //指向当前访问的结点指针
}Linklist;
Linklist L1, L2, L3,L4,result;
多项式链表的节点和链表类型

Status InitList(Linklist &L)
{
	L.head = (Link)malloc(sizeof(LNode));
	if (!L.head)
		exit(OVERFLOW);
	else
	L.head->next= NULL;
	L.tail = L.head;
	return OK;
}//分配空初始化链表
```

```
Status creatlist(Linklist &L, int n)
{
	Link newnode;
	L.head= (Link)malloc(sizeof(LNode));
	if (!L.head)
		return ERROR;
	  L.tail = L.head;
	  L.tail->next = NULL;
	  ;
	for (int i = 0;i < n;i++)
	{
		newnode= (Link)malloc(sizeof(LNode));
		if (!newnode)
			return ERROR;
		else
		{
			cin >> newnode->coef >> newnode->exp;
			L.tail->next = newnode;
			L.tail = newnode;
		}
	}
	L.tail->next = NULL;
	return OK;
}//尾插法创建链表，存入数据
Status insertafter(Link &s, int a, int b)
{
	Link p;
	  p= (Link)malloc(sizeof(LNode));
	if (!p)
		return ERROR;
	else
	{
		p->coef = a;
		p->exp = b;

		if (s->next == NULL)
		{
			s->next = p;
			s = p;
			s->next = NULL;
		}
		else
		{
			p->next = s->next;
```

```
                s->next = p;
            }
        }
        return OK;
}//在指定节点后插入数据
void sum(Linklist &L1, Linklist &L2,Linklist &L3)
{
        Link p1, p2,p;
        p1 = L1.head->next;
        p2 = L2.head->next;
        while (p1 && p2)
        {
            if (p1->exp == p2->exp)
            {
                int t = p1->coef + p2->coef;
                if (t != 0)
                    insertafter(L3.tail, t, p1->exp);
                p1 = p1->next;
                p2 = p2->next;
            }
            else if (p1->exp < p2->exp)
            {
                insertafter(L3.tail, p1->coef, p1->exp);
                p1 = p1->next;
            }
            else if (p1->exp > p2->exp)
            {
                insertafter(L3.tail, p2->coef, p2->exp);
                p2 = p2->next;
            }
        }
            if (p1 != NULL)p = p1;
            else p = p2;
            while (p != NULL)
            {
                insertafter(L3.tail, p->coef, p->exp);
                p = p->next;
            }
}//两多项式链表执行加法操作
void cheng(Linklist &L1, Linklist &L2)
{
        Link p1 = L1.head->next;
        Link p2 = L2.head->next;
        Link pi;
```

```
            InitList(result);
             while (p2 != NULL)//L1 的第一个节点和 L2 相乘 遍历 L2,结果存入
result
            {
                  /*pi = (Link)malloc(sizeof(LNode));
                  if (!pi)
                        exit(OVERFLOW);*/
                       insertafter(result.tail,p2->coef*p1->coef, p2->exp +
p1->exp);
                  p2 = p2->next;

            }
            pi = (Link)malloc(sizeof(LNode));
            if (!pi)
                  exit(OVERFLOW);
            p1 = p1->next;
            Link q =result.head;
            pi = result.head->next;
            //L1 第一个节点之后的每一个节点与 L2 每一项相乘，按顺序插入 result
            while (p1 != NULL)
            {
                  p2 = L2.head->next;
                  while (p2)
                  {
                        pi = result.head->next;
                        q = result.head;
                       while ( pi!=NULL&& pi->exp < (p1->exp + p2->exp))
                     {
                        pi = pi->next;
                        q = q->next;
                       }
                       if (pi != NULL) {
                              if (pi->exp == (p1->exp + p2->exp))
                              {
                                    pi->coef += p1->coef * p2->coef;
                              }
                              if (pi->exp < (p1->exp + p2->exp))
                              {
                                          insertafter(result.tail, p1->coef *
p2->coef, p1->exp + p2->exp);
                              }
                              if (pi->exp > (p1->exp + p2->exp))
                              {
```

```
                                    insertafter(q, p1->coef * p2->coef,
p1->exp + p2->exp);
                        }
                    }
                    else {
                        if (q->exp == (p1->exp + p2->exp) ) {
                            q->coef += p1->coef * p2->coef;
                        }
                        if (q->exp < (q->exp + q->exp))
                        {
                                    insertafter(result.tail, p1->coef *
p2->coef, p1->exp + p2->exp);
                        }
                    }

                     /*if (q->exp == (p1->exp + p2->exp) && pi == NULL) {
                         q->coef += p1->coef * p2->coef;
                     }
                     if (pi->exp > (p1->exp + p2->exp))
                     {
                          insertafter(q, p1->coef * p2->coef, p1->exp +
p2->exp);
                     }
                     if (pi->exp < (p1->exp + p2->exp))
                     {
                          insertafter(result.tail, p1->coef * p2->coef,
p1->exp + p2->exp);
                     }*/
                     p2 = p2->next;
                }
                     p1 = p1->next;
            }
}//两多项式链表执行乘法操作
```

2.3.5 调试与分析

本程序写的过程中出现了不少问题，比如在加法时每次找到相同指数执行加法后，没有执行指针 p->p.next 操作，最后是采用单步调试查看数据和执行情况发现了漏洞并及时补上，在乘法过程中需要用到插入操作，但是一开始写的插入操作是头插法导致数据输出是反序的，多试了几次分析结果后就找到了问题所在。总的来说乘法操作应该还可以再优化一下，因为本代码提交 oj 时显示后几个测试点超过了 timelimit 说明时间复杂度比较高，执行了太多次插入和遍历操作，根据一些参考书，可以采用先找到最高次数，再逐项相乘的方式进行优化。

2.3.6 总结和体会

链表的操作比较注重细节，某几个操作的顺序发生变化可能导致程序功能发生变化，同

时在写程序的过程中我也详细了解熟悉了头插法建表和尾插法建表的区别所在。熟悉了基本的链表操作，但是对于多项式乘法这种稍微复杂一些的问题，我能想到的还是基于加法的比较朴素的操作，导致算法效率也相对较低，得分没有满，希望能在后面的学习中开拓思路，熟悉代码操作，提升算法能力和效率。

2.4 求级数

2.4.1 问题描述

利用高精度的加法和乘法实现求级数的操作

2.4.2 基本要求

输入：

输入若干行，在每一行中给出整数 N 和 A 的值，（1<=N<=150，0<=A<=15）

对于 20%的数据，有 1<=N<=12，0<=A<=5

对于 40%的数据，有 1<=N<=18，0<=A<=9

对于 100%的数据，有 1<=N<=150，0<=A<=15

输出：

对于每一行，在一行中输出级数 A+2A2+3A3+。。。+NAN =$\sum_{i=1}^{N} iA^i$的整数值

2.4.3 数据结构设计

由于可以预测有一大部分数据超过了 64 位二进制的上限，如果直接作乘法与加法的话会超出上限产生截断，得到错误结果，于是我们需要用顺序表模拟加法与乘法操作，并将结果的每一位数存入顺序表中再输出。本题比较方便的方法是采用 string 类型，由于 string 类储存的数据位数远大于此题结果的位数，因此不会产生越界。构造两个函数，一个进行加法操作，一个进行乘法操作，最后再根据要求的级数进行循环执行加法和乘法，得到级数结果。其中加法操作是通过模拟竖式进位的方式记录每位相加的进位数，逐位相加，乘法也是采用模拟竖式乘法的方式，再把结果前面的 0 全部除去。

2.4.4 功能说明（函数、类）

```
string add(string str1, string str2)
{
	string str;
	int len1 = str1.length();
	int len2 = str2.length();
	if (len1 < len2)
	{
		for (int i = 1;i <= len2 - len1;i++)
			str1 = "0" + str1;
	}
	else
	{
		for (int i = 1;i <= len1 - len2;i++)
			str2 = "0" + str2;
	}
	len1 = str1.length();
```

```
	int jinwei = 0;
	int temp;
	for (int i = len1 - 1;i >= 0;i--)
	{
		temp = str1[i] - '0' + str2[i] - '0' + jinwei;
		jinwei= temp / 10;
		temp %= 10;
		str = char(temp + '0') + str;
	}
	if (jinwei != 0)   str = char(jinwei + '0') + str;
	return str;
}//高精度加法
string mul(string str1, string str2)
{
	string str;
	int len1 = str1.length();
	int len2 = str2.length();
	string tempstr;
	for (int i = len2 - 1;i >= 0;i--)
	{
		tempstr = "";
		int temp = str2[i] - '0';
		int t = 0;
		int cf = 0;
		if (temp != 0)
		{
			for (int j = 1;j <= len2 - 1 - i;j++)
				tempstr += "0";
			for (int j = len1 - 1;j >= 0;j--)
			{
				t = (temp * (str1[j] - '0') + cf) % 10;
				cf = (temp * (str1[j] - '0') + cf) / 10;
				tempstr = char(t + '0') + tempstr;
			}
			if (cf != 0) tempstr = char(cf + '0') + tempstr;
		}
		str = add(str, tempstr);
	}
	str.erase(0, str.find_first_not_of('0'));//去0
	return str;
}//高精度乘法
```

2.4.5 调试与分析

此题由于是将每一个数都看成 string 类型，因此涉及到 string 类和 int 类转换，开始时由于没有进行转换，导致报错或者结果出错，后来一方面可以用 int temp = str1[i]

-'0'和 to_string 函数进行相互转换，数据的使用便方便了很多。还有一个问题在于处理加法和乘法的进位数时，容易出现语句顺序不对导致结果出错的问题，在画图模拟竖式运算步骤后也得到了解决。

2.4.6 总结和体会

此题高精度暗示我们不能直接相加相乘，要用线性表进行模拟运算操作，这也正是难点所在，其实本题不用 string 类，就用普通的线性表也一样可以实现，不过 string 类在乘法后面去 0 等等操作上具有便捷度上的优势性。此题看似简单，却花费了我不少时间，而且很长时间是属于代码写得快但是 bug 找的慢，自认为问题还是基本代码不太熟练，细节注意的不到位，调试能力有待提高。需要平时多加努力。

2.5 扑克牌游戏

2.5.1 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。

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。

对于 20%的数据，n<=20，有 Append、Pop 指令

对于 40%的数据，n<=50，有 Append、Pop、Revert 指令

对于 100%的数据，n<=200，有 Append、Pop、Revert、Extract 指令

2.5.2 基本要求

输入：第一行输入一个整数 n，表示命令的数量。

接下来的 n 行，每一行输入一个命令。

输出：

输出若干行，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（花色和数字），若牌堆为空则输出 NULL

2.5.3 数据结构设计

本题看似复杂，要求却十分明确，思维含量不大，建立顺序表，主要执行四个操作即可。可以规定顺序表表头为牌堆底部，表尾为排队顶部。其中反转即为将顺序表逆序排列，添加即每次插入一个元素至表头，弹出就是先执行判断非空操作再删除表尾元素。Length--；

Extract 操作相对比较复杂，首先需要定义扑克牌点数的大小优先级。我这里使用的是 map 函数映射十三张牌点数的大小顺序，方便进行排序和判断。首先遍历原顺序表，遇到指定花色便复制到另一个顺序表 B 中，再将 B 中所有元素按大小优先级进行排序，排序完成后，再一次遍历原顺序表，利用 append 操作从表尾

开始，只要不是指定花色的牌便插入表头，这样最后得到的顺序表 B 便是要求的顺序表。
判断指令功能时，利用 strcmp 函数判断是否输入某一特定指令，进而执行相应的操作即可。

2.5.4 功能说明（函数、类）

```
map<string, int> priority;//定义 map 函数
typedef int Status;
typedef struct card {
	char a[5];
	char color[10];
	/*card() {
		color = (char*)malloc(10);
	}*/
}card;
typedef struct {
	card* x;
	int length;
	int listsize;
}cardlist;
cardlist L;//定义顺序表的元素类型，顺序表类型

Status Initlist(cardlist& L)
{
	L.x = (card*)malloc(LIST_INIT_SIZE * sizeof(card));
	if (!L.x)exit(OVERFLOW);
	L.length = 0;
	L.listsize = LIST_INIT_SIZE;
	return OK;
}//分配储存空间并初始化顺序表
Status append(cardlist& L, char m[], char n[])
{
	card* p, * q, * newbase = NULL;
	if (L.length == 0)
	{
		strcpy_s(L.x[0].a, m);
		//q=&(L.x[0]);
		/*while (n != NULL)
		{
			*(q->color) = *n;
			n++;
			(q->color)++;
		}*/
		strcpy_s(L.x[0].color, 10, n);
		/*L.x[0].color = n;*/
```

```
		L.length++;
	}
	else {

		if (L.length >= L.listsize)
		{
			newbase = (card*)realloc(L.x, L.listsize + 100);
		}
		q = &(L.x[0]);
		for (p = &(L.x[L.length - 1]);p >= q;--p)
			*(p + 1) = *p;
		strcpy_s(q->a, m);
		/*q->color = n;*/
		/*while (n != NULL)
		{
			*(q->color) = *n;
			n++;
			q->color++;
		}*/
		strcpy_s(q->color, 10, n);
		++L.length;
	}
	return OK;
}//执行在表头插入操作，判断是否内存不够，如果内存不够需要额外分配内存
cardlist extract(char s[], map<string, int> priority)
{
	cardlist C;
	Initlist(C);
	for (int i = 0;i < L.length;i++)
	{
		if (strcmp(L.x[i].color, s) == 0)
		{
			append(C, L.x[i].a, L.x[i].color);
		}
	}
	//sort
	for (int i = 1;i < C.length;i++)
	{
		for (int j = 0;j < C.length - i;j++)
		{
			string a = C.x[j].a;
			string b = C.x[j + 1].a;
			if (priority[a] > priority[b])
			{
```

```
                        card temp = C.x[j];
                        C.x[j] = C.x[j + 1];
                        C.x[j + 1] = temp;
                  }
            }
      }
      for (int i = L.length - 1;i >= 0;i--)
      {
            if (strcmp(L.x[i].color, s) != 0)
            {
                  append(C, L.x[i].a, L.x[i].color);
            }
      }
      return C;
}//提取指定花色的牌，放入新线性表中，执行排序操作，再将原表中非指定花色的牌倒序插入新表表头
void revert(cardlist& L)
{

      card temp;
      for (int i = 0;i < L.length / 2;i++)
      {
            temp = L.x[i];
            L.x[i] = L.x[L.length - 1 - i];
            L.x[L.length - 1 - i] = temp;
      }
}//将顺序表倒置
void pop(cardlist& L)
{
      if (L.length == 0)
            cout << "NULL" << endl;
      else {
            card* p = &(L.x[L.length - 1]);
            cout << p->color << " " << p->a << endl;
            --L.length;
      }
}//判断表是否空，如果非空删除表尾元素
```

2.5.5 调试与分析

本题要实现的功能较多，结构比较复杂，但是总体的思路还是比较清晰的，在编程过程中，起初因为没有考虑点数为 10 是两个字符，直接把 card 类型里的点数设为 char 类型，导致输入 10 时只能读入 1 而编译没有报错，导致错误，后来将其改为 char[]类型，同时将 map 函数的映射类型改成从 string 类型映射到 int 类型。还有就是要注意题目要求，题目要求最后倒序输出，在 extract 操作要求从小到大排列后放在排堆顶，这时候要分清楚牌堆顶和牌堆底，不能弄错了顺序。开始在内存上也存在一些问题，通过单

步调试，发现 malloc 只是分配内存，没有执行默认的析构函数和构造函数，如果一开始在 card 类型中设为 char*而不对其初始化的时候，使用过程中可能会出错。

2.5.6 总结和体会

对于这类功能较多的问题，需要一部分一部分的实现，将每一部分封装好，使某一部分的功能不会受到其他部分出错的影响，这样程序整体的健壮性会更好。我还需要多进行这类编程练习，在查找 bug 中不断提升自己的代码能力。

3.实验总结

线性表是数据结构中最基础的部分，其中的一些基本操作的思想也会贯彻到后面的学习中去，这次实验通过具体的题目让我们更加熟悉了线性表的各种操作，提高了相关代码的熟练度，加深了对这种数据结构的理解。

此次作业是我进入大数据专业接触数据结构课程的第一次实验，不论是从完成的时间还是完成的质量上都有一些瑕疵，包括一元多项式的乘法部分仍然应该算法效率不够高没有拿到满分。在编程的过程中，代码的实现有些小细节我仍然处理的不到位，并且代码整体的熟练度不高，容易出现一些逻辑上的问题。调试能力也需要加强，在完成过程中往往出现写代码很快但是改代码改很久的情况，一方面要提升找错误的能力，一方面在写的时候也要注意细节，减少漏洞。