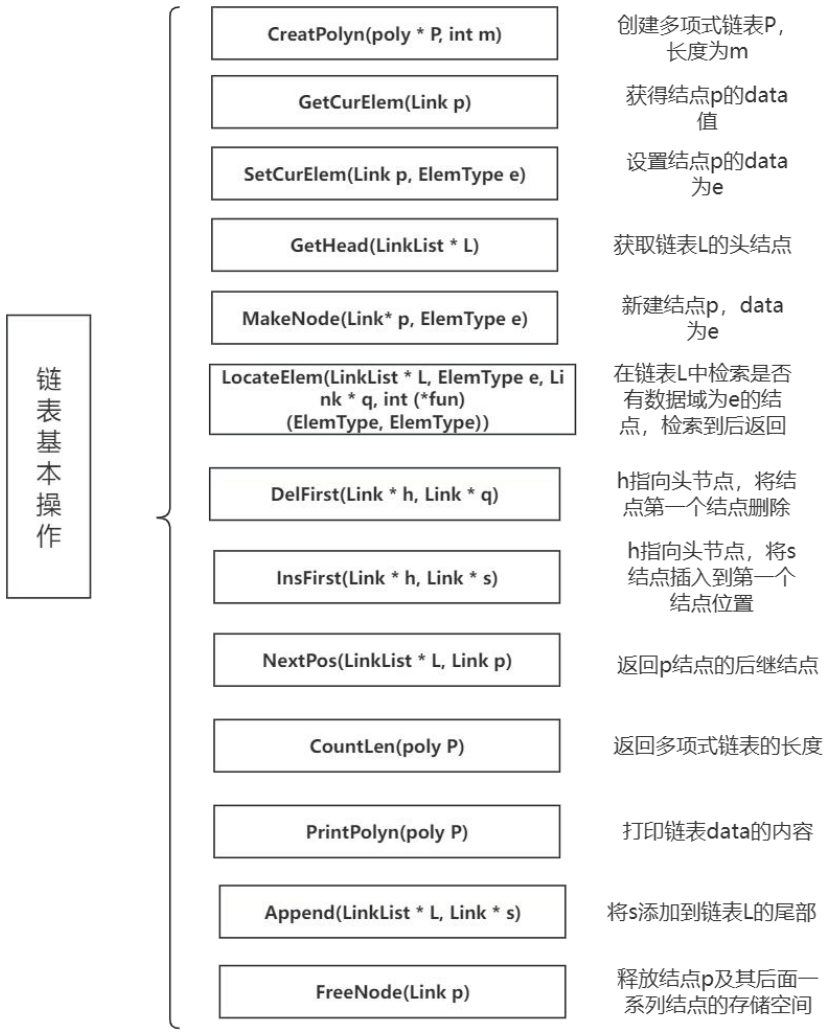
作业 HW1* 实验报告

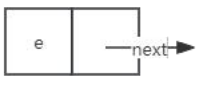
姓名：朱俊泽 学号：2351114 日期：2024 年 10 月 3 日

实验报告格式要求按照模板（使用 Markdown 等也请保证报告内包含模板中的要素）
对字体大小、缩进、颜色等不做强制要求（但尽量代码部分和文字内容有一定区分，可参考 vscode 配色）
实验报告要求在文字简洁的同时将内容表示清楚
报告内不要大段贴代码，尽量控制在 20 页以内

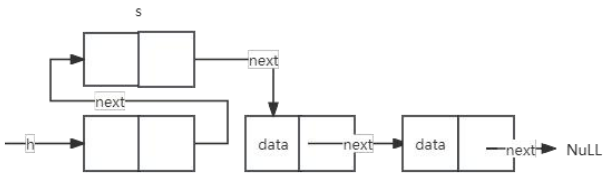
1. 涉及数据结构和相关背景

顺序表 链表 # 题目或实验涉及数据结构的相关背景

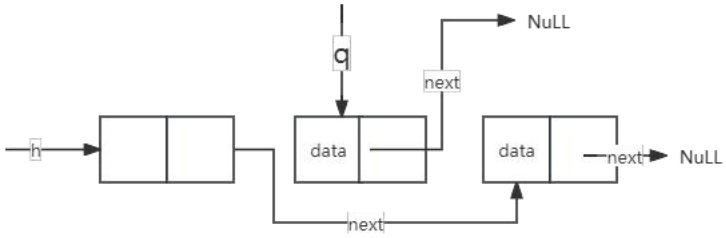




创建节点（创建头结点）



插入：
把前一个节点的下一个指针指向自己，把自己的下一个指针指向原本前一个指针指向的（如果是尾插那就省略）



删除：
把前一个节点的下一个指针指向自己的下一个指针，然后释放自己的内存（尾插可以直接指向 null 然后释放）

2. 实验内容

2.1 轮转数组

2.1.1 问题描述

给定一个整数顺序表 nums，将顺序表中的元素向右轮转 k 个位置，其中 k 是非负数

2.1.2 基本要求

输入第一行两个整数 n 和 k，分别表示 nums 的元素个数 n，和向右轮转 k 个位置；
第二行包括 n 个整数，为顺序表 nums 中的元素

输出轮转后的顺序表中的元素

2.1.3 数据结构设计

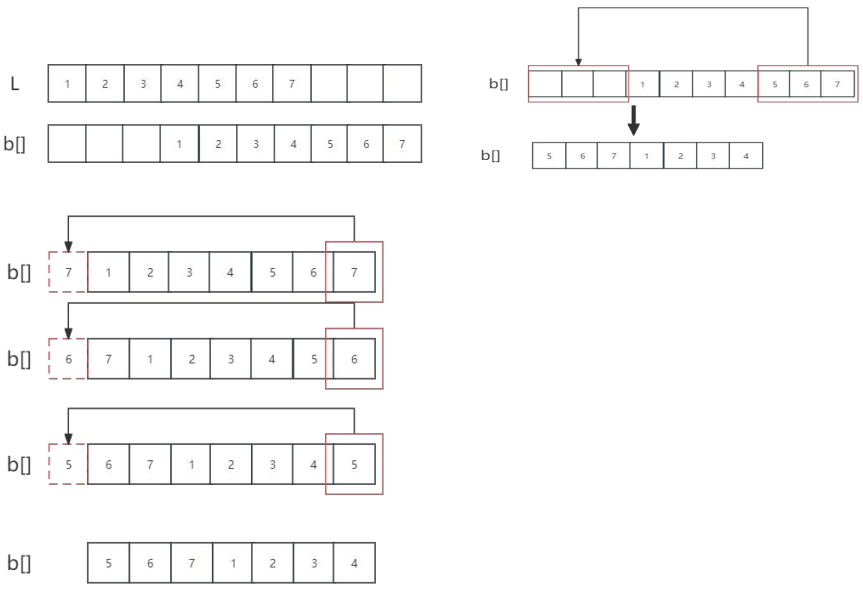
```cpp
int a[100005];
void solve()
{
    int n,k;
    cin>>n>>k;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
    }
    k=k%n;
    for(int i=n-k+1;i<=n;i++)
    {
        cout<<a[i]<<" ";
    }
    for(int i=1;i<=n-k;i++)
    {
        cout<<a[i]<<" ";
    }
}
```

一个简单的线性数组，因为轮转具有循环性

2.1.4 功能说明（函数、类）

此时按照轮转的结束点输出即可

2.1.5 调试分析（遇到的问题和解决方法）



这样就可以发现循环性

2.1.6 总结和体会

注意好边界的 k=k%n 这样就可以避免轮完了一圈

2.1.7 链表方法

```
//定义线性表的结构
typedef struct
{
    ElemType* data;//顺序表元素
    int length;//顺序表当前长度
}SqList;
```

```
int SqList_Init(SqList& L)
{
    L.data = new ElemType[Maxsize];//为顺序表分配一个大小为Maxsize的数组空间
    L.length = 0;//空表长度为0
    return 0;
}
```

存储数据 data，记录顺序表长度 length。初始化线性表

至于刚刚实现的尾部到头部的交换，可以如下：

```
void swap(int* a, int* b, SqList& L) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
void Bubble(SqList& L,int k,int n) {
    while (k--) {
        for (int i = n - 1; i > 0; i--) {
            swap(&(L.data[i - 1]), &(L.data[i]), L);
        }
    }
}
```

可以通过交换值实现

2.2 一元多项式的相加和相乘

2.2.1 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。本题输入保证是按照指数项递增有序的。

对于%15 的数据，有 1<=n,m<=15

对于%33 的数据，有 1<=n,m<=50

对于%66 的数据，有 1<=n,m<=100

对于 100%的数据，有 1<=n,m<=2050

本题总分 150 分，得到 100 分或以上可视为满分

2.2.2 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 个整数，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 个整数，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

2.2.3 数据结构设计

```
int ar2[MAX_DEF]= {0};
int ar1[MAX_DEF]= {0};
int ADDar[MAX_DEF] = {0};
int MULar[DOU_MAX]= {0};
```

注意题目给出的两个一元多次不等式具有递增性质，这样会很好处理
暂时不需要定义不连续的链表来节省存储空间，定义四个连续的数组空间就足够了
数据范围指数上就算相乘也是 2e3+2e3 也是 4e3 的数量级

但是如果数据量再扩大的话，就需要定义一个不连续的链表来实现空间的节省了

2.2.4 功能说明（函数、类）

```
void Add(int a1[],int a2[],int b[]) {
    for(int i=0; i<1001; i++)
        b[i]=a1[i]+a2[i];
}
```

相加的函数比较简单，对应位次相加即可：
阶数

1	A1	→	1	0		1	A1
2	0	→	2	0		2	0
3	A2	→	3	B1		3	B1+A2
4	0	→	4	0		4	0
5	0	→	5	0		5	0
6	0	→	6	0		6	0
7	A3	→	7	B2		7	B2+A3
...	...						

```
void Mul(int a1[],int a2[],int b[]) {
    for(int i=0; i<1001; i++) {
        for(int j=0; j<1001; j++) {
            b[i+j]+=a1[i]*a2[j];
        }
    }
}
```

相乘的函数注意对应的阶数是相加，然后底数是相乘：
阶数

1	A1	→	1	0	→	1	0
2	0	→	2	0	→	2	0
3	A2	→	3	B1	→	3	0
4	0	→	4	0	→	4	0
5	0	→	5	0	→	5	0
6	0	→	6	0	→	6	0

7	A3		7	B2		7	
...	...						

图示只是模拟了第一个数组的第一阶乘第二个数组，注意好指数的相加即可

```
void OutPut(int ar[],int range) {
    int p=0;
    for(int i=0; i<=range; i++) {
        if(ar[i]!=0) {
            if(p)putchar(' ');
            printf("%d %d",ar[i],i);
            p=1;
        }
    }
    if(!p)printf("0 0");
}
```

输出就比较简单，注意题目要求的 0 0 输出

2.2.6 总结和体会

注意题目说的 0 1 2 三个判断和 0 0 输出
另外注意设置好边界量，否则会在运行的时候提醒访问出错。

2.2.7 链表做法

```
//Link里存的是指针的地址
typedef struct LNode {//结点类型
    ElemType data;//数据域
    struct LNode* next;//指针域
}*Link;//Link存放的是每个结点的地址
```

```
//定义节点中数据域的数据类型
typedef struct {
    double p;//系数
    int e;//指数
}ElemType;
```

```
//空链表中只有一个头指针，头指针中存储着一个节点的地址。这个节点就是头结点
typedef struct {//链表类型
    Link head;//指向线性链表中的头结点
    int len;
}LinkList;
typedef LinkList poly;
```

定义好链表的 data 数据类型，还有链表

然后

```
    case 0://a==b，两者的指数值相等
        e.e = a.e;
        e.p = a.p + b.p;
        //修改多项式Pa中当前结点的系数值
        if (e.p != 0.0) {    //相加不为零更新Pa节点对应的数据
            SetCurElem(qa, e);
            ha = qa;
        }
        else {              //相加为则直接删除该结点
            DelFirst(&ha, &qa);
            FreeNode(qa);

        }
        DelFirst(&hb, &qb);
        FreeNode(qb);
        qb = NextPos(Pb, hb);
        qa = NextPos(Pa, ha);
```

实现多项式相加

```
qb = NextPos(&Pb,hb);//初始化qb，qb在计算中指向Pb中的某一项
while (qb)    //模拟数学计算的分配律，Pb逐项与Pa整个多项式相乘
{
    ha = GetHead(&Pa);
    qa = NextPos(&Pa,ha);
    InitList(&Pc);        //Pc是辅助链表，保存Pb的其中一项与Pa相乘的结果
    while (qa)            //qb指向的项与qa中的每一项依次相乘
    {
        e.p = qa->data.p * qb->data.p;
        e.e = qa->data.e + qb->data.e;
        MakeNode(&s, e);//Pc一开始是一个空链表，相乘结果存到一个新结点中，再添加到Pc里
        Append(&Pc, &s);
        ha = qa;
        qa = NextPos(&Pa, ha);
    }

    AddPolyn(&Pd, &Pc);       //Pd也是一个新链表
    hb = qb;
    qb = NextPos(&Pb,hb);//qb步进，指向Pb中的下一项
    free(hb);
}
```

实现多项式相乘

在计算完多项式后，需要进行合并和排序！！！

```
while (q) { //将链表中结点数据域中的指数两两比较（时间复杂度O(n^2)）
    if (q != NULL) {  //q指向一个结点，p指向q后的某一个结点，h为辅助指针，指向p前紧邻的节点，作为虚拟头结点作为删除后一个结点的参考
        a = GetCurElem(q);
        p = NextPos(P, q);
        h = q;
        while (p) {
            b = GetCurElem(p);
            if (a.e == b.e) { //一列中遇到两个指数相等的
                a.p = a.p + b.p;
                SetCurElem(q, a);
                //p此时指向待删除的结点，h为p紧邻的前一个节点
                DelFirst(&h, &s);
                len--;
            }
            h = p;
            p = NextPos(P, p);
        }
    }
    q = NextPos(P, q);//q步进到下一个结点
}
```

```
for(int i = 1; i <=len-1; i++){
    h = GetHead(P);
    q = NextPos(P, h);//q指针指向前结点
    p = NextPos(P, q);//p指针指向后结点
    for (int j = 1; j <= len - i ; j++){
        a = GetCurElem(q);
        b = GetCurElem(p);

        if (a.e > b.e) {    //若前结点中的指数比后结点的指数大，则需要相互交换
            temp= GetCurElem(q);
            SetCurElem(q, p->data);
            SetCurElem(p, temp);
        }
        q = p;
        p = NextPos(P, p);   //p，q指针同时步进到下一个结点，从而进行下一次比较
    }
}
```

2.3 求级数

2.3.1 问题描述

做一个高精度求级数

2.3.2 基本要求

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

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

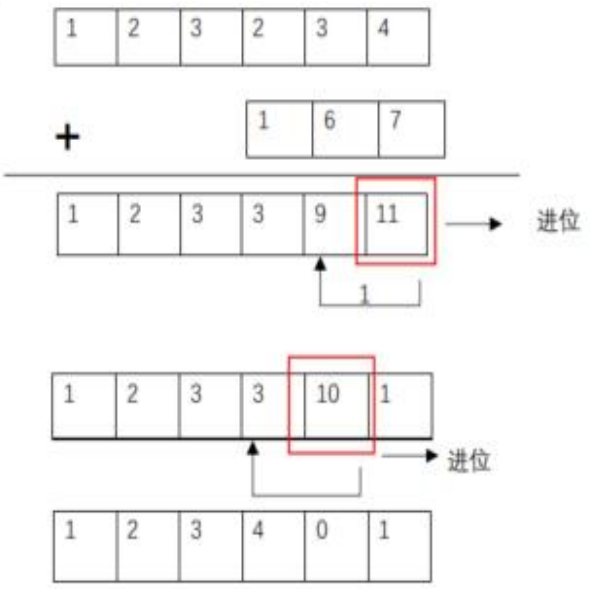
2.3.3 数据结构设计

模拟一个 string 高精度，使用 char[]数组即可

原先使用 string 完成的高精度操作，模拟一个 char[]数组完成即可，数组大小可以使用 maxlen 结合数据范围进行估计

2.3.4 功能说明（函数、类）

加法实际是对齐的每一位分别相加，所得结果对每一位再进位处理。
进位过程：除 10，结果为进位位数，加到高位，余数为该位保留的数。

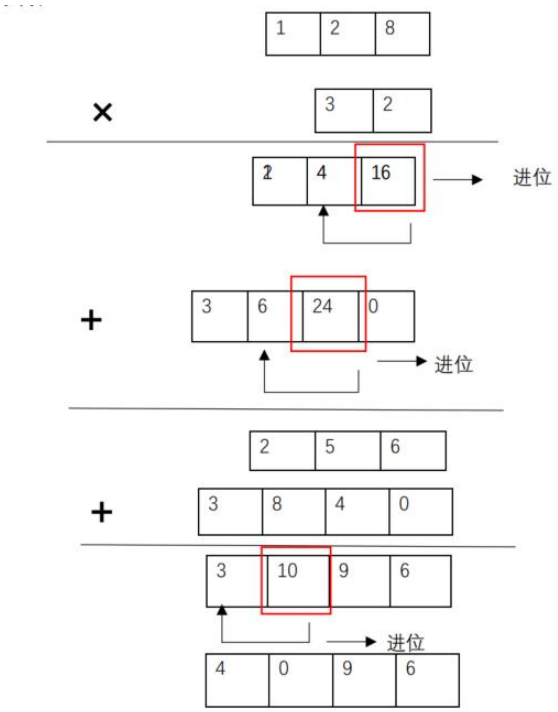



```
string gjd_add(string str1,string str2)//高精度加法
{
    string str;
    int len1=str1.length();
    int len2=str2.length();
    //前面补0，弄成长度相同
    if(len1<len2)
    {
        for(int i=1;i<=len2-len1;i++)
          str1="0"+str1;
    }
    else
    {
        for(int i=1;i<=len1-len2;i++)
          str2="0"+str2;
    }
    len1=str1.length();
    int cf=0;
    int temp;
    for(int i=len1-1;i>=0;i--)
    {
        temp=str1[i]-'0'+str2[i]-'0'+cf;
        cf=temp/10;
        temp%=10;
        str=char(temp+'0')+str;
    }
    if(cf!=0)  str=char(cf+'0')+str;
    return str;
}
```

乘法操作时，进行的是每位对应两两相乘，第 i 位和第 j 位相乘，结果放到(i+j)位，进位后的得到的多个数按照加法法则再相加。

图为 string 的高精度加法，cf 用于控制进位，%10 就是本位留下来的位次，/10 就是进给下一位次的。然后加到最高位的时候记得判断此时 cf 进位是不是 0

如果是 0 就结束，不是 0 的话还需要进一位




```
string gjd_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=gjd_add(str,tempstr);
    }
    str.erase(0,str.find_first_not_of('0'));
    return str;
}
```

图为 string 下的高精度，之后转化成 char[]数组的高精度，乘法其实就是执行了多个位置的多次加法，当本位置不为'0'的时候，用来和对位每一个位置*之后%10 就是本位剩下的，然后/10 就是进给下一位的，用 cf 来保存

Char[]中加上了 jinwei 来保存模拟进位，其他的逻辑基本一致

2.2.6 总结和体会

本题用到了高精度算法，但是难点并不在于如何构造高精度，而在于如何正确逆序存放好输入的数据以及中间运算量，并将其倒序正确输出

2.4 扑克牌游戏

2.4.1 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。

2.4.2 基本要求

输入：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
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

2.4.3 数据结构设计

```
const int N = 250;
struct card
{
    string col;//花色
    string num;//号码
};
card a[N];
int top=0;
```

链表要是一个有顺序的表：（双端）
使用 top 来记录现在的最顶端

Card[1]
Card[2]
Card[3]
Card[4]
Card[5]
‘’’’
Card[N]

2.4.4 功能说明（函数、类）

```
if(op=="Append")
{
    top++;
    for(int i=top;i>=2;i--)
    {
        if(i==1)
            break;
        a[i].col=a[i-1].col;
        a[i].num=a[i-1].num;
    }
    cin>>a[1].col>>a[1].num;
}
```

把所有的节点往后移动一个，然后输入 a[1]的内容

```
else if(op=="Pop")
{
    if(top==0)
        cout<<"NULL"<<endl;
    else
    {
        cout<<a[top].col<<" "<<a[top].num<<endl;
        top--;
    }
}
```

抽走 top 位置的内容

```
else if(op=="Revert")
{
    for(int i=1;i<=top/2;i++)
    {
        card temp=a[i];
        a[i]=a[top-i+1];
        a[top-i+1]=temp;
    }
}
```

遍历从第一个到 top/2 个，实现交换即可

```
else if(op=="Extract")
{
    string dir;
    cin>>dir;
    int temptop=0;
    int Temp[100];
    memset(Temp,0,sizeof Temp);
    for(int i=1;i<=top-temptop;i++)
    {
        if(a[i].col==dir)
        {
            temptop++;
           if (a[i].num==  "A")  Temp[1]++;
            if (a[i].num==  "2")  Temp[2]++;
           if (a[i].num==  "3")  Temp[3]++;
           if (a[i].num==  "4")  Temp[4] ++;
         if (a[i].num==  "5")  Temp[5]++;
            if (a[i].num==  "6")  Temp[6] ++;
           if (a[i].num==  "7")  Temp[7] ++;
         if (a[i].num==  "8")  Temp[8] ++;
           if (a[i].num== "9")  Temp[9] ++;
           if (a[i].num== "10") Temp[10]++;
           if (a[i].num==  "J")  Temp[11] ++;
           if (a[i].num== "Q")  Temp[12] ++;
           if (a[i].num== "K")  Temp[13] ++;
           //cout<<"num"<<a[i].num<<endl;
```

实现抽出所有同颜色卡牌的时候，全部抽出，用一个 temp[]数组来记录抽走了那些纸牌，然后从高到底遍历 temp[]因为要求是这么放入牌堆。

2.4.6 总结和体会

本题除了使用顺序表编写，也可用链表实现。本题借助顺序表实现，在执行 reverse 反向指令上效率更高，因为顺序表可以随机访问表中的元素。另外本题在编写顺序表功能的时候可以考虑功能可拓展性。如题目中要求的是删除牌堆顶元素以及添加到牌堆顶端，但是可以直接编写插入到任意位置的功能，方便维护和灵活修改

2.4.7 链表方法

```
//定义顺序表内容
typedef struct {
    ElemType* elem;//elem数组指针指向当前线性表的基址
    int  length;//线性表当前长度
    int  listsize;//指示顺序表分配的空间的大小
    Card card[301];
}SqList;
```

定义好链表内容，然后元素类型依然不变。一个颜色一个点数

```
L.length++;
Card* p = (&L.card[i]);//首指针
Card* m = (&L.card[L.length]);//尾指针
for (; p <= m; m--) {
    *(m + 1) = *m;
}
cin >>L.card[i].Decor >> L.card[i].ID;
```

插入一个元素

```
L.length--;
Card* p = (&L.card[i]);
Card* m = (&L.card[L.length]);
for (; p <= m; p++) {
    *p = *(p + 1);
```

删除元素

```
void RevertList(SqList& L) {
    int i = 1;
    int j =L.length;
    Card temp;
    for (; i <= L.length / 2; i++, j--) {
        temp = L.card[i];
        L.card[i] = L.card[j];
        L.card[j] = temp;
    }
}
```

逆转

```
for (int j = 1; j <= i; j++) {
    SqListInsertInput(L, j, L_Decor, ID_Store[j]);//将ID_Store[]中的内容依次放到牌堆
}
```

```
//冒泡排序
for (int i = 1; i <= k - 1; i++) {
    for (int j = 1; j <= k - i; j++) {
        if (Temp[j] >= Temp[j + 1]) {
            t = Temp[j];
            Temp[j] = Temp[j + 1];
            Temp[j + 1] = t;
        }
    }
}
```

和原本的思想一下使用 temp[]进行排序和映射来实现抽取

2.5 学生信息管理

2.5.1 问题描述

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。

本题 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。

2.5.2 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,字符串长度<100)
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。

注：全部数值 <= 10000，元素位置从 1 开始。 学生信息有重复数据（输入时未做检查），查找时只需返回找到的第一个。
每个操作都在上一个操作的基础上完成。

2.5.3 数据结构设计

学生结构体:

```
struct Pupil {
    int serialNum;
    string identifier;
    string fullName;
};
```

链表:

```
struct SequentialList {
    ElemType* elements;
    int currentLength;
    int allocatedSize;
    Pupil students[100001];
};
```

2.4.4 功能说明（函数、类）

```
bool InitializeList(SequentialList& list) {
    list.elements = (int*)malloc(INITIAL_LIST_SIZE * sizeof(int));
    list.currentLength = 0;
    list.allocatedSize = INITIAL_LIST_SIZE;
    return true;
}
```

初始化一个顺序表，先分配好空间然后设置当前长度，之后把分配的长度设置给list.allocatedSize

```
void InputToList(SequentialList& list, int numStudents) {
    for (int i = 1; i <= numStudents; ++i) {
        list.currentLength++;
        list.students[i].serialNum = i;
        SAFE_CIN(list.students[i].identifier);
        SAFE_CIN(list.students[i].fullName);
    }
}
```

从输入中读取学生信息，并添加到顺序表的学生数组中。

```
void InsertToList(SequentialList& list, int position) {
    if (position < 1 || position > list.currentLength + 1) {
        cout << "-1" << endl;
    } else {
        if (list.currentLength <= MAX_LIST_LENGTH) {
            int* newSpace = (ElemType*)realloc(list.elements, (list.allocatedSize + LIST_INCREMENT) * sizeof(ElemType));
            list.elements = newSpace;
            list.allocatedSize += 1;
        }

        for (int i = list.currentLength; i >= position; --i) {
            list.students[i + 1] = list.students[i];
        }

        list.currentLength++;
        list.students[position].serialNum = position;
        SAFE_CIN(list.students[position].identifier);
        SAFE_CIN(list.students[position].fullName);

        cout << "0" << endl;

        for (int i = 1; i <= list.currentLength; ++i) {
            list.students[i].serialNum = i;
        }
    }
}
```

在指定位置插入一个新学生。
检查插入位置是否合法（1 到 currentLength + 1 之间）。
如果当前长度未超过最大限制，则尝试扩展内存空间。
将插入位置后的所有学生后移一位。
插入新学生，并更新其信息。

```
void DeleteFromList(SequentialList& list, int position) {
    if (position < 1 || position > list.currentLength) {
        cout << "-1" << endl;
    } else {
        for (int i = position; i < list.currentLength; ++i) {
            list.students[i] = list.students[i + 1];
        }

        list.currentLength--;
        cout << "0" << endl;


        for (int i = 1; i <= list.currentLength; ++i) {
            list.students[i].serialNum = i;
        }
    }
}
```

从指定位置删除一个学生
判断位置合法性
将删除位置后的所有学生前移一位。
更新当前长度
更新所有学生的序列号

```
void QueryByID(SequentialList& list, string query) {
    bool found = false;
    for (int i = 1; i <= list.currentLength; i++) {
        if (list.students[i].identifier == query) {
            cout << list.students[i].serialNum << " " << list.students[i].identifier <
            found = true;
            break;
        }
    }
    if (!found) {
        cout << "-1" << endl;
    }
}
```

遍历学生数组，查找与给定标识符匹配的学生
如果找到，输出学生的完整信息
如果未找到，输出 "-1"

```
void QueryByName(SequentialList& list, string query) {
    bool found = false;
    for (int i = 1; i <= list.currentLength; i++) {
        if (list.students[i].fullName == query) {
            cout << list.students[i].serialNum << " " << list.students[i].identif
            found = true;
            break;
        }
    }
    if (!found) {
        cout << "-1" << endl;
    }
}
```

遍历学生数组，查找与给定名字匹配的学生
如果找到，输出学生的完整信息
如果未找到，输出 "-1"

3. 实验总结

本次实验的核心内容是线性表，总共包括顺序表和链表两类。两者都很经典的线性存储的数据结构，但在存储特点、修改方面的效率都有不同，适用于不同场合。其中顺序表的内存空间是连续的，且可以直接随机访问表中的任意一个序号的元素，但是在表中间删除、插入等方面需要借助循环覆盖，运行效率低，在表尾增删效率高；此外顺序表中经常会出现空余空间，空间利用效率低。总的来看可以当做可以存储任意数据类型元素的数组使用。适用于高频次访问表中的元素的任务中如本次实验中**题目轮转数组**、**学生信息管理**而链表的内存空间不连续的，有可能会出现内存破碎的问题，而空间利用效率高，在表中就占内存，不在表中的结点可以直接释放内存空间，且在表中插入、删除数据只需要简单调整 next 指针指向。但是不支持直接随机访问任意位置的数据，需要从头结点开始依次向后遍历。链表适用于需要经常扩充缩减表内容的任务，如**题目一元多项式的相加和相乘。**而对于**题目扑克牌游戏**，借助链表与顺序表均可以实现，两者各有优势。因为题目中所有的添加和删除都是在牌堆底部，两者效率都很高。但是在 reverse 反向操作中，由于顺序表可以直接随机访问

元素，因此顺序表实现效率更高；在 extract 中，涉及到多个中间元素的删除，因此链表效率更高。由此可见具体选择哪种数据结构要充分考量任务的特点。此外本次任务还涉及到了简单的算法设计。**题目求级数**涉及到了高精度算法，本质是模拟自然运算的进位思想，将C++可存储、运算的数据范围进一步扩大。另外通过此次实验，也认识到将各个功能通过函数、类单独封装成可调用的接口可以大大提高代码的可读性并且方便后期维护，在定义大量的函数和类的时候，要特别关注全局变量、局部变量以及动态表动态变量的使用，以免因为变量的范围不同导致程序出错。**相比较使用数组**，我发现使用**链表**在动态维护方面优秀很多