实验一：合并有序数组

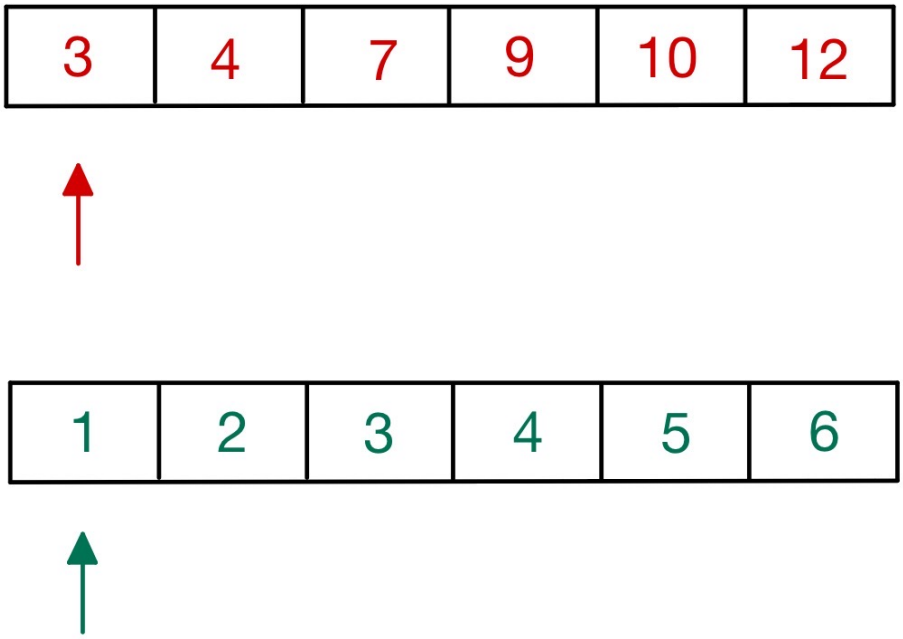
给定两个有序数组，请把它们合并成一个升序数组并输出。

问题分析

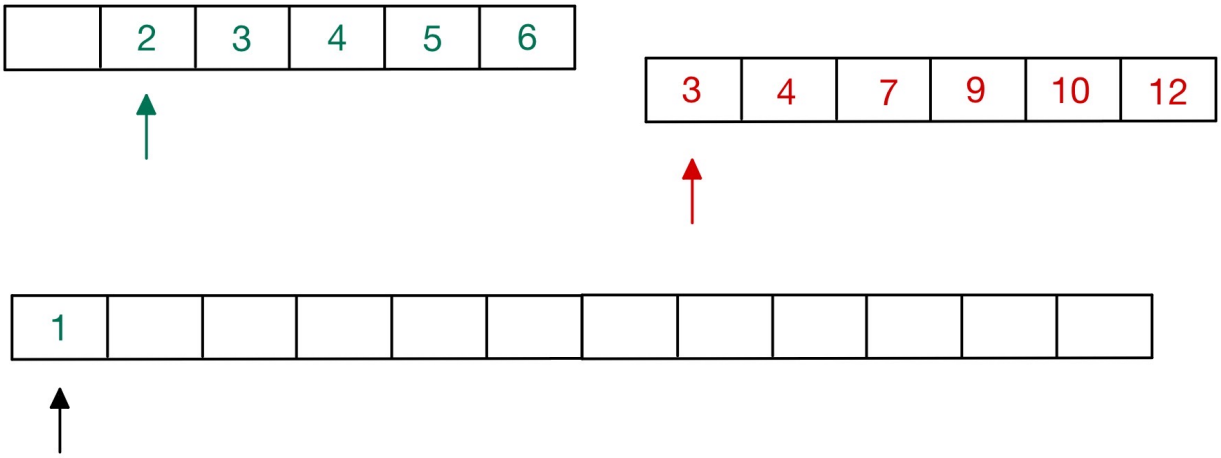
提供两个升序数组，要求仍然合并为升序数组，可以用归并的思想求解

概要设计

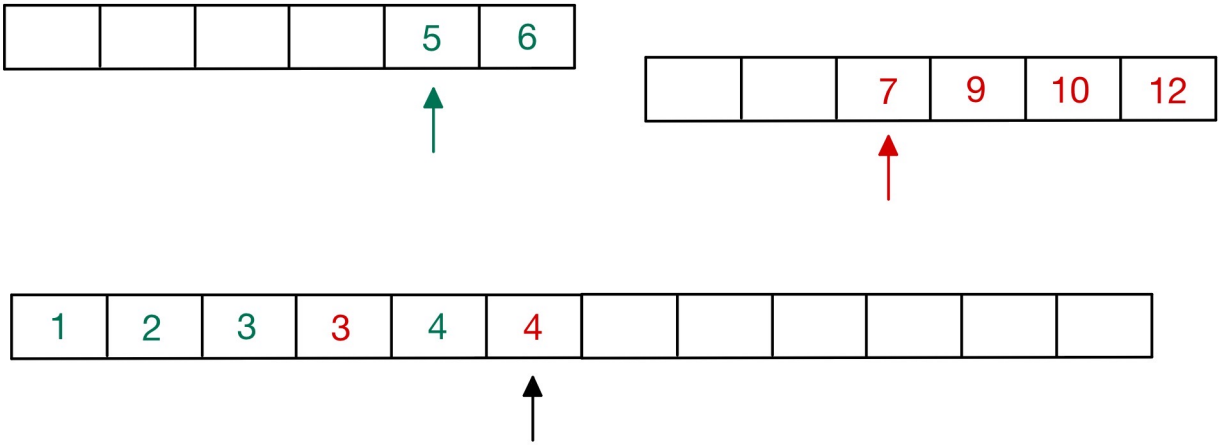
设计两个“指针”，分别从两个数组的头部开始



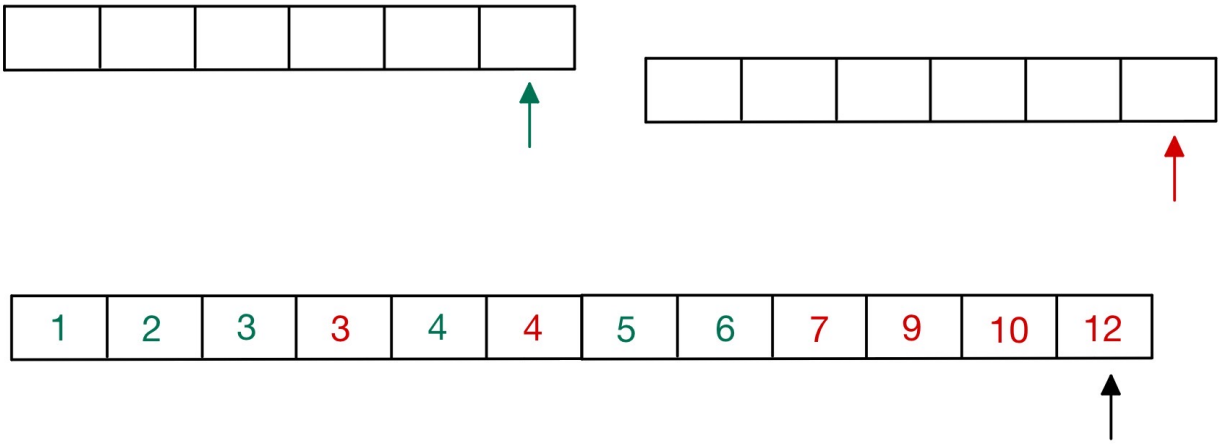
比较两个指针对应值的大小，把小的那个归并到结果数组，对应指针后移一位。当然，我们还需要一个指针来记录该数值在结果数组中的存储位置



重复此过程，我们遇到一个问题————如果找到一个值，同时存在于两个源数组中，按照题目的语义，应该在结果数组中**删去**一个，保证合并结果是一个**升序数组**，而不是一个**不降数组**，二者在数学上有本质的区别。但是在实际编写的过程中，noj并没有通过删去后的数组，而是默认两个源数组中的相同数字均被放入结果数组。*我认为这是一个bug，在此提出*。



当指针指向源数组的末尾时，不再增加，当两个指针都在各自数组的末尾时，结束程序，得到结果。



详细设计

元素设计及输入

```
int n,m;//源数组长度
int x=0,y=0,z=0;//源数组、结果数组的指针
int A[300],B[300],C[400];//源数组、结果数组 开大一点防止奇怪错误
//输入
scanf("%d",&n);
for(int i=0;i<n;i++) scanf("%d",&A[i]);
scanf("%d",&m);
for(int j=0;j<m;j++) scanf("%d",&B[j]);
```

归并循环体

```
while(!(y==m&&x==n))//两个指针同时到达尾部时停止循环
{
    if(y==m||x<n&&A[x]<=B[y]) //当B数组指针到底，或者A当前元素小于等于B当前元素时，将A指针进行取数后移操作。
    {
       C[z]=A[x]; //取数
        x++; //后移
    }
    else if(x==n||y<m&&A[x]>=B[y])
    {
       C[z]=B[y];
        y++;
    }
   z++;
```

```
}
```

输出

```
for(int s=0;s<z-1;s++)
{
    printf("%d\n",C[s]);
}
printf("%d\n",C[z-1]); //防止oj抽风
```

完整代码见源文件 Noj_ex001_arrayMerge.c

程序使用及测试结果

使用方法：按照oj要求进行输入

1. 第一行为第一个有序数组的长度，正整数n，n<=20；
2. 第二行为第一个有序数组的n个数字，用空格隔开；
3. 第三行为第二个有序数组的长度，正整数m，m<=20；
4. 第四行为第二个有序数组的m个数字，用空格隔开。
   运行截图，测试了一些极端情况，通过oj。

```
(base) yuanhaocheng@yuanhaochengdeMacBook-Pro noj_ex % ./001
0
1
1
1
```

```
1
(base) yuanhaocheng@yuanhaochengdeMacBook-Pro noj_ex % ./001
3
1 2 3
4
1 2 3 4
1
1
2
2
3
3
4
```

实验二：高精度计算PI值

限制使用双向链表作存储结构，请根据用户输入的一个整数（该整数表示精确到小数点后的位数，可能要求精确到小数点后500位），高精度计算PI值。可以利用反三角函数幂级展开式来进行计算。

问题分析

提示我们使用反三角函数的幂级数展开式，我们考虑反正弦函数

$$f(x) = arcsin(x)$$

将其展开，我们有

$$f(x)=\sum_{n=0}^{\infty}\frac{n!}{(2n+1)!}x^{2n+1}$$

注意这里要求$|x|<1$

所以对于π，我们有

$$\begin{aligned}\pi&=2*\frac{\pi}{2}\\&=2arcsin(1)\\&=2\sum_{n=0}^{\infty}\frac{n!}{(2n+1)!}\end{aligned}$$

我们记$R(n)=\frac{n!}{(2n+1)!}$，故而有

$$R(n)=\frac{n}{2n+1}R(n-1)\quad(1)$$

于是有

$$\pi=2\sum_{n=0}^{\infty}R(n)\qquad(2)$$

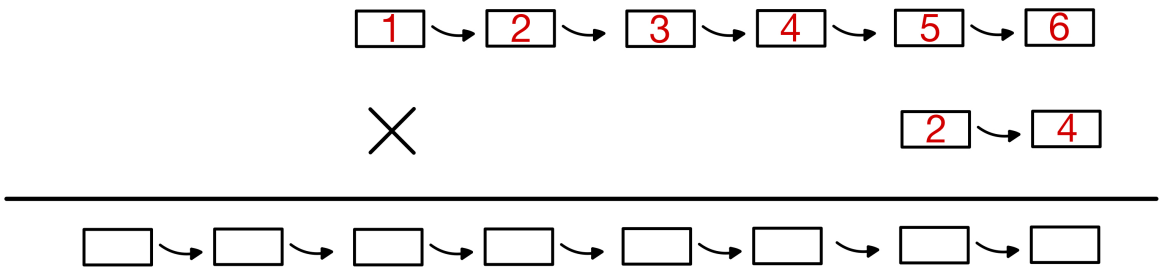
利用这个公式，可以高精度计算π值，但是由于所需精度大，自带的数据类型不能完整表示，采用链表模拟手算的思路进行大数运算。

概要设计

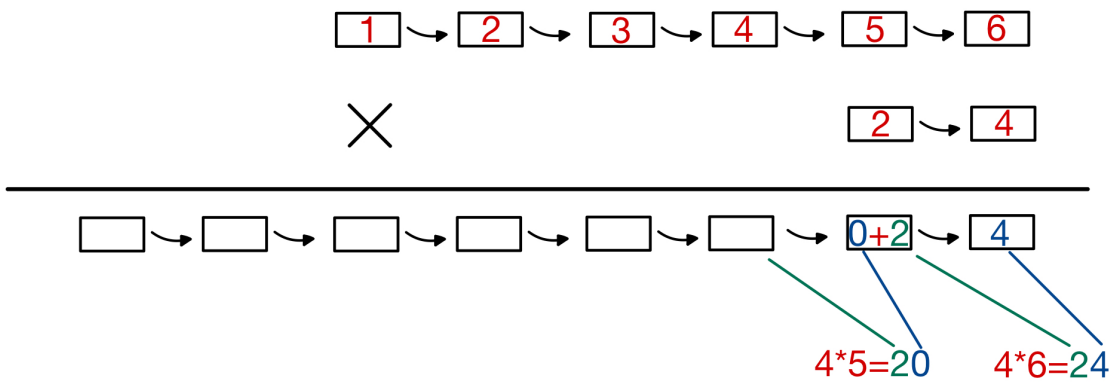
主体是一个循环，利用公式$(1)(2)$重复计算足够多次（本例计算3000次），得到高精度的π值。

过程中涉及大数运算，基本思路是，将运算理解为竖式计算，用链表存储结果，举一例大数乘法如下：

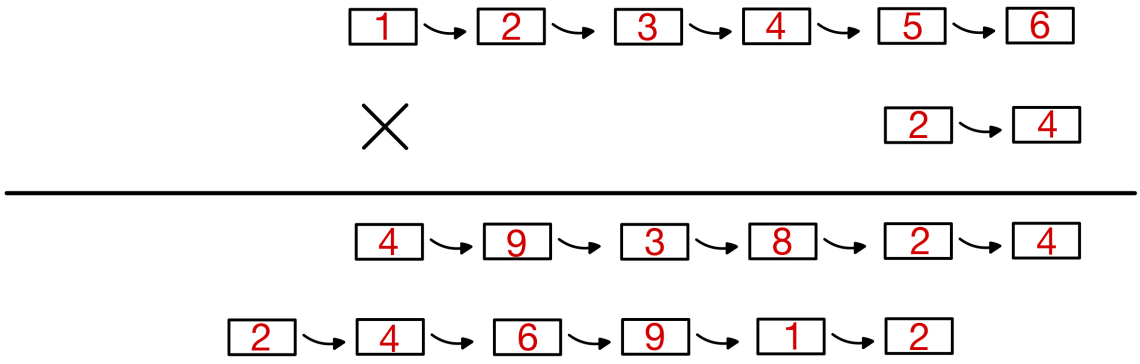
乘数与被乘数



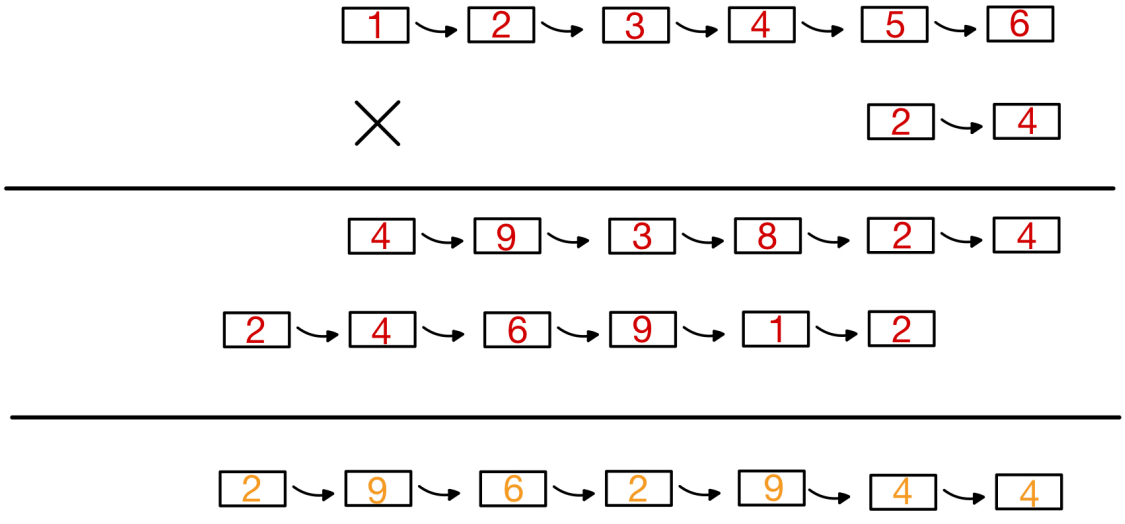
计算个位与被乘数的乘积，将得数模10记录在对应位，除以10记录为进位



对乘数各位重复操作



得数相加



这是算法的思路，在实际操作中，由于存储限制，我们不必逐个算出每一行的得数，可以在最终得数的链表里，从低位向高位计算，这样可以节约空间。

其余大数运算类似

详细设计

元素设计及初始化

```
struct Node{ //单个节点
    char elem;
    Node *nex, *pre;
    Node(int elem = 0, Node *nex = 0, Node *pre = 0):elem(elem), nex(nex),
pre(pre){}
};
typedef Node *LinkList;

struct BigDec{ //大数运算链表，所有的运算都是对这个链表的操作，因此重载了加乘除（用不到减法）
    LinkList Head;
    BigDec(int n = 500){
        int i = n;
        LinkList L,s;
        L = this->Head = new Node();
        L->nex = L->pre = L;
        while(i--){
            s = new Node(0, L->nex, L);
            L->nex->pre = s;
            L->nex = s;
        }
    }
    BigDec operator * (const long long rsh){
    ......
    }
    BigDec operator / (const long long rsh){
        ......
    }
    BigDec operator + (const BigDec &rsh){
    ......
    }
};
```

大数运算模块

大数乘法

```
BigDec operator * (const long long rsh){ //rsh是乘数 默认this链表为被乘数
        LinkList pa= this->Head->pre;
        long long s = 0;
        while(pa != this->Head){ //遍历每一位
            long long x = rsh*(pa->elem);
            pa->elem = (x+s)%10; //模10记录在本位
            s = (x+s)/10; //除10记录在进位
            pa = pa->pre;
        }
        this->Head->elem += s;
        return *this;
    }
```

大数除法

```
BigDec operator / (const long long rsh){
        LinkList pa = this->Head->nex;
        long long s = this->Head->elem;
        if(s<rsh) {this->Head->elem = 0;s = (s%rsh)*10;}
        else {this->Head->elem = s/rsh; s = (s%rsh)*10;}
        while(pa != this->Head){
            s += pa->elem;
            if(s < rsh ){ pa->elem = 0;s = (s%rsh)*10;pa = pa->nex;}
            else {pa->elem = s/rsh;s = (s%rsh)*10; pa = pa->nex;}
        }
        return *this;
    }
```

大数加法

```
BigDec operator + (const BigDec &rsh){
        LinkList pa,pb;
        pa = this->Head->pre;
        pb = rsh.Head->pre;
        int s = 0;
        while(pa!=this->Head){
            int x = (pa->elem + pb->elem);
            pa->elem = (x+s)%10;
            s = (x+s)/10;
            pa = pa->pre;pb = pb->pre;
        }
        return *this;
    }
```

计算π

```
int N; //N为精度
scanf("%d", &N);
BigDec *a = new BigDec(N+10);
BigDec *b = new BigDec(N+10);
a->Head->nex->elem = 5; //初始化为5
*b+*a; //b记录结果
long long x, y;
for(int i = 1; i<=N*10; i++){
        x = (long long)(2*i-1)*(2*i-1);
        y = (long long)(2*i)*(2*i+1)*4;
        (*a)*x;
        (*a)/y;
        //BigDec_pri(*a);
        (*b)+(*a);
    }
```

输出

```
void BigDec_pri(BigDec &BDec, int n){//打印链表
    LinkList p = BDec.Head->nex;
    printf("%d.", BDec.Head->elem);
    while(n--){
        printf("%d", p->elem);
        p = p->nex;
    }
    printf("\n");
}
```

完整代码见源文件 Noj_ex001_countPI.cpp

程序使用及测试结果

程序通过noj测试，计算精度为500时结果如下：

```
(base) yuanhaocheng@yuanhaochengdeMacBook-Pro noj_ex % ./0022
500
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912
```

实验心得及体会

上学期已经学过了算法设计与分析，因此本次实验的难度并不大，但在实际编写代码时，总有一些考虑不周到的情况发生。

链表是一个重要的数据结构，但并不是一个非常好用的模版，事实上，操作指针给编写程序带来了较大的风险，个人而言，还是倾向于使用数组或者封装好的$vector$类代替链表。

实验二实际上可以使用各种公式进行计算，条条大路通罗马。

代码规范是重要的，它能给调试带来很大的方便。

一次AC很难，要心态平和。